REPORT DOCUMENTATION PAGE

AFRL-SR-BL-TR-01-

0462

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	February 2001	Final Technical Report, 4/15/98-11/14/00

4. TITLE AND SUBTITLE
New World Vistas: Planning and Scheduling
Planning and Scheduling with Contingencies

5. FUNDING NUMBERS
F49620-98-C-0023

6. AUTHOR(S)
Michael Curry
Prof. David A. Castañon

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
ALPHATECH, Inc.
50 Mall Rd.
Burlington, MA 01803

8. PERFORMING ORGANIZATION REPORT NUMBER
TR-1005

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Dr. Neal Glassman
Air Force Office of Scientific Research
110 Duncan Avenue, Room B115
Bolling Air Force Base, DC 20332-8080

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

20010905 129

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited

13. ABSTRACT *(Maximum 200 Words)*

Planning and scheduling have been identified as one of several critical technology areas required to achieve information superiority. In order to exploit future information systems that provide information in real-time, commanders must have fast real-time techniques to modify plans in the presence of uncertainty, and which anticipate future replanning so that the original plans can be modified with minimal impact.

The proposed research was to develop mathematical techniques for fast mission replanning in a class of risky multiplatform assignment and scheduling problems. The mathematical models represented multiple platforms evolving on graphs, with risky transitions where platforms can be destroyed, and task nodes that platforms can perform. The resulting Markov decision problem can be solved exactly for small problems using stochastic dynamic programming. To extend these algorithms to larger dynamic decision problems, we developed different approximations to the stochastic dynamic programming algorithm, and explored their relative performance on classes of test problems. The results identified advantages and potential shortcomings of several techniques, and identified promising techniques for application to Air Force planning and scheduling problems. These techniques will have the capability of generating plans and schedules which anticipate contingencies, and which provide for efficient replanning in cases of failures.

14. SUBJECT TERMS
Approximate Dynamic Programming, Rollout Algorithms, Neural Networks, Multi-vehicle Routing with Risk, Planning with Contingencies, Monte Carlo Evaluation, Markov Decision Problem

15. NUMBER OF PAGES
158

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFED	UNCLASSIFED	UL

TR-1005

Rev. 1.0

FINAL TECHNICAL REPORT

NEW WORLD VISTAS: PLANNING AND SCHEDULING

DYNAMIC ASSIGNMENT AND SCHEDULING WITH CONTINGENCIES

Contract F49620-98-0023

CDRL Item 0002AA

14 February, 2001

Submitted by:

ALPHATECH, Inc.

50 Mall Road,

Burlington, MA 01803-4562

Submitted to:

Dr. Neal Glassman

Air Force Office of Scientific Research

110 Duncan Avenue, Room B115

Bolling Air Force Base, DC 20332-8080

Table of Contents

1 ACKNOWLEDGEMENTS

Contributors to this program were Prof. David Castañon, Prof. Dimitri Bertsekas, Prof. Stephen Patek, David Logan (PM), Michael Curry (PL), and Dr. Cynara Wu.

2 INTRODUCTION AND MOTIVATION

In *Joint Vision 2010* [1], the Chairman of the Joint Chiefs of Staff, General John Shalikashvili, outlined a vision of effective, efficient armed services for the next century. In this document, General Shalikashvili stressed the importance of "information superiority" which he defined as "The capability to collect, process, and disseminate an uninterrupted flow of information while exploiting or denying an adversary's ability to do the same." Enhanced command, control, and intelligence provided by information superiority will transform the traditional functions of maneuver, strike, protection and logistics into a new conceptual framework for operations. The new operational concepts are called dominant maneuver, precision engagement, full dimensional protection, and focused logistics. These concepts will exploit information superiority and other technological advances to out-maneuver any adversary, attack with precision, defend friendly forces, and maintain supplies to widely distributed assets.

In a subsequent report, entitled *New World Vistas* [2], the Air Force Scientific Advisory Board identified specific technology areas that must be developed to achieve information superiority and to maintain the Air Force as the preeminent air force of the twenty-first century. One of the critical technology areas was Scheduling and Planning, focusing on the development of efficient methodologies for distributed planning and scheduling it exploits information superiority provided by new Air Force systems.

Planning and scheduling problems arise in many important Air Force applications such as Intelligence, Surveillance and Reconnaissance (ISR), sensor management, and strike planning and replanning. Such applications provided the motivation and focus for our research. In these applications, limited surveillance, logistics and combat resources are carefully scheduled to best achieve a set of missions. However, information regarding the battle space contains uncertainty, is incomplete, is not always correct, and evolves continuously; furthermore, many of these problems involve potential actions by adversaries which affect mission success. A major advantage of future information systems is that they will provide the capability for close monitoring of mission progress and the evolution of the battle space, and thus provide the opportunity for effective, timely replanning to increase mission success.

In order to illustrate the nature of the problems addressed in the research, we summarize three motivating Air Force applications below.

2.1 ISR

ISR problems involve the planning and adaptive replanning of surveillance platform trajectories (e.g. U2, Global Hawk, etc.) and scheduling of mission assignments to examine and monitor important areas in the battle space. The diversity of available platforms and sensors that is envisioned in future combat systems would provide commanders with a greater capability and flexibility in selecting and scheduling their intelligence, surveillance, and reconnaissance missions. The objective of ISR mission plan is to generate coordinated multiplatform collection schedules, which combine synergistically the sensor capabilities of the different platforms to detect, track and identify the enemy components in the battle space. The outputs of the mission plans include specification of the routes of each platform (e.g., its ground track and altitude), the targets/areas that each platform must examine along its route, and the schedule of data collection activities for each platform.

In the event that a platform is destroyed, or new objects are detected in the battle space, the missions of the individual platforms must be modified to maintain coverage of the important events, while accommodating new tasks. Algorithms, which can perform such real-time replanning in anticipation and response to unpredictable events, were the focus of our research.

2.2 SENSOR MANAGEMENT

Sensor management consists of scheduling the activities of diverse, geographically located mobile sensors in order to obtain an accurate classification of objects in the battle space. This class of problems is very similar to the ISR problem; the difference is that sensor management occurs at a faster time scale. ISR is concerned about platform trajectories, and responds to events such as new target detections and platform destruction. In contrast, sensor management is focused on sensor mode selection, and responds to real-time tracking and classification information. From a planning perspective, sensor management focuses on the concept of managing the information dynamics provided by the sensors. As information is acquired, targets that have a higher likelihood of being a threat should be examined more closely in order to accurately determine their type, whereas objects which pose little expected threat should not be viewed. This feedback management of information provides the type of information superiority envisioned in *Joint Vision 2010.*

2.3 STRIKE PLANNING AND REPLANNING

The last class of motivating applications is the planning, scheduling and subsequent replanning of air combat missions in offensive strike operations. In the simplest variation of these planning and scheduling problems, individual platforms are assigned routes and missions for each route, and dynamically adapt both routes and missions in response to observed contingency events such as the discovery of new enemy threats or targets, or the destruction of some platforms. More complex versions of these problems involve planning a set of integrated missions, together with logistic, surveillance, and electronic support assets; such planning must anticipate future contingencies which may change mission assignments, and be able to respond with real-time replanning. Events such as detection of new threats, failure to destroy intended targets, and loss of platforms require modifying the existing set of missions in order to accomplish the intended objectives.

In the remainder of this report, we describe the progress we made towards the development of planning and scheduling algorithms, which anticipate the need for future replanning and thus generate robust missions which are readily modified when unpredicted contingencies arise.

3 RESEARCH APPROACH

The planning and scheduling problems outlined above have the following characteristics in common:

1. A series of decisions must be made over time (the original plan plus revisions due to replanning).
2. Each decision has an immediate cost or payoff that must be considered and has consequences that affect future decisions and hence future costs or payoffs.
3. Information about the effects of decisions is observed (by monitoring the progress of the plans and contingencies), and can be used to modify future decisions.
4. The immediate and long-term consequences of a decision may be influenced by random (uncertainty) and deliberate (antagonist) factors beyond the control of the decision maker.

Thus, the above problems are a class of dynamic decision problems under uncertainty. One of the first results of our research was the development of a mathematical paradigm to represent the important aspects of the applications discussed previously. This led to a class of problems denoted as multi-vehicle routing problems on risky graphs. The modeling paradigm is described in greater detail in Section 3 below and in the papers in the appendix. The resulting optimization was a Markov decision problem, which could be solved in principle by the mathematics of stochastic dynamic programming (SDP) [3]. However, the required state space representation of the problem grew exponentially with the size of the problem description, making direct application of the SDP algorithm impossible except for very simple problem instances.

Our research objectives were to develop algorithms, which approximated the performance of the SDP algorithm, while maintaining feasible computation requirements for large problems. In order to do this; we abandoned the control-theoretic model of computing a priori an optimal feedback strategy for every

possible future state of the system. Instead, we adopted the philosophy of receding horizon control and model-predictive control, whereby a set of control decisions is computed in real-time, by solving a replanning problem starting from the current state. In this manner, we considered states that could be reached from the current state, instead of all possible states. In order to compute the decisions made at the current state; a receding horizon decision problem was formulated starting at the current state, and was solved in real-time, using techniques which approximated the SDP algorithm.

Our research explored a number of variations of this approach, focusing on promising techniques that have been used for computing highly effective, if not provably optimal, solutions to stochastic, dynamic optimization problems. These techniques are based on a combination of ideas from operations research, control, and artificial intelligence. The common elements of these techniques were:

- Efficient approximation of the cost-to-go function from dynamic programming.
- Local Search for approximate minimization of current cost plus expected cost-to-go.

Specifically, our research investigated how to develop effective cost-to-go approximations for problems with large state and decision spaces. We explored the use of neural network approximations (as in neurodynamic programming [4]), simulation-based methods (the temporal rollout techniques of [5-7]), plus decomposition techniques and other approximations. The various techniques are described in the papers enclosed in the Appendices, along with the results of the different evaluation experiments we performed.

In the next section, we summarize the results of our research effort, much of which has been published and included as attachments.

4 RESEARCH RESULTS

4.1 MODEL PROBLEM

In this section, we describe a mathematical formulation that represents the important aspects of multi-platform scheduling under risk. This formulation served as the basic focus for our research. The paradigm used in the formulation is that platforms are scheduled to perform spatially distributed tasks.

Our model starts with a finite set of discrete nodes N, which represent potential locations of tasks or intermediate waypoints enroute to task locations. Each node n is characterized by a value $V(n)$ of the task associated with that node. This value can vary dynamically, and is reduced to zero if the task at that node has already been performed, or if there is no task at the node.

We assume that there is a set of directed arcs A connecting the nodes N. Associated with each of arc a is a risk $p(a)$, which represents the probability that a platform traversing that arc will not be destroyed on that arc, and thus will reach the end node of the arc.

Platforms travel on the associated graph $G(N,A)$. There are M identical platforms that are present in the graph. We describe the evolution of each platform in terms of a discrete time index k. We assume that each platform can traverse a single arc in a unit of time. Although this assumption implies that each arc can be traveled in unit time, this assumption is not restrictive because additional nodes can be introduced to represent waypoints, which are equally spaced. Associated with each platform m at time k is a platform state $s_k(m)$, which is either -1 to indicate the platform has been destroyed, or indicates the node which contains the platform at time k.

The state of the system at time k is defined as x_k, and is composed of the collection of individual platform states and node value states. Let $v_k(n)$ denote a binary state for node n at time k, which is 1 if no platform has reached the node before or at time k, and 0 otherwise. Then, the system state is defined by the vector:

$$x_k^T = [v_k(1), \ldots, v_k(N), s_k(1), \ldots s_k(M)]^T$$

The admissible decisions at time k, denoted by $U(x_k)$, depend on the current state x_k. Each platform m which is not destroyed at time k (so $s_k(m) > 0$) must select an arc in A to traverse which starts at node $s_k(m)$; staying at the current node is not allowed unless there is an arc in A which starts and ends at that node. When platform m traverses an arc $a = (s_k(m), e)$ at time k, its state changes as follows:

$$s_{k+1}(m) = e \text{ with probability } p(a); \text{ otherwise, } s_{k+1}(m) = -1.$$

We assume that the stochastic events associated with each platform traversing an arc are independent across platforms, arcs and time. Thus, each arc traversal represents an independent Bernoulli event that affects the state of an individual platform.

The node value dynamics are deterministic; and represented as follows: If a platform reaches a node, its node value is automatically reduced to 0. With this notation, the state dynamics can be represented as

$$x_{k+1} = f_k(x_k, u_k, \omega_k)$$

where x_k is the state, u_k is the control to be selected from a finite set $U_k(x_k)$, and ω_k represents the random events associated with the arc transitions by platforms.

The final aspect of the model is the objective function. We formulate the decision problem as a finite horizon problem, with maximum time T. We assume that, at time $k=0$, all platforms start at a base node $n=0$. We are interested in all platforms returning to base by time T. Associated with each platform is a platform value V_m, which is lost if the platform does not safely return to base. With this notation, the net value at the final time T when the system reaches state x_T is given by the sum of task value accumulated minus platform value lost, as

$$J(x_T) = \sum_{n \in N} V(n) I(v_T(n) = 0) + \sum_{m=1}^{M} V_m I(s_T(m) = 0)$$

We can regroup the above performance in terms of the incremental value accumulated at each time, as follows:

$$J = \sum_{k=1}^{T} \{ \sum_{n \in N} V(n) I(v_{k-1}(n) = 1, v_k(n) = 0) \} + \sum_{m=1}^{M} V_m I(s_T(m) = 0) = \sum_{k=0}^{T-1} g_k(x_k, u_k, \omega_k) + G(x_T)$$

where $I()$ is the indicator function. The objective J is random since it depends on the outcomes of the random arc traversal events. We assume that, at each state k, the state x_k is observed, and the choice of control action u_k depends on the current state x_k.

Although the above formulation assumes that all platforms are identical, it is straightforward to include multiple types of platforms, by making the arc transition probabilities depend on platform type j, as $p(a,j)$. It is also straightforward to make the value of a node depend on the time at which a platform reaches the node, as $V(n,k)$, thereby incorporating constraints such as windows of time during which tasks are available at nodes. A third straightforward extension is to restrict the type of platform that can perform the task at a specific node. Although we do not explicitly treat such examples in this paper, the methodology presented below extends naturally to those cases.

The above model is a Markov decision problem, with observed state x_k. Note, however, that the number of possible different states is $2^V(V+1)^M$, where V is the number of nodes in the graph. Thus, the number of states explodes rapidly with the number of nodes. This makes infeasible the computation of a full feedback strategy, which selects an action for every possible state and time.

In the remaining subsections, we summarize the results in our major publications. These are included in the Appendices.

4.2 ROLLOUT ALGORITHMS FOR STOCHASTIC SCHEDULING PROBLEMS

The first set of investigations focused on the special case in the previous model where there is a single platform present in the problem. For this case, when the network graph is fully connected and the arc risks depend only on the terminal node, the problem reduces to a well-known stochastic scheduling problem known as a quiz problem [7]. This class of problems has a known optimal solution, which consists of an index policy, which ranks nodes in order of desirability. The optimal policy for fully connected graphs is to visit the nodes in the order provided by the index policy.

We focused on extensions of the problem to cases with sparse graphs, limited time horizons, precedence constraints and additional uncertainty For this class of problems, determination of optimal decision rules required the solution of large-scale stochastic dynamic programming problems. In our research, we developed a new class of algorithms, based on approximate dynamic programming principles, which achieved near optimal performance while requiring far less computation.

The main idea behind our approximate dynamic programming algorithm was to avoid computation of an optimal control for every possible state. The premise of dynamic programming is to precompute such an optimal policy, thereby requiring complete exploration of all of the possible state realizations and events which may happen. The concepts we explored adopted the philosophy of model-predictive control, where computation of a decision waits until the current state is observed. In this manner, one can restrict the future states considered to those states, which are likely to be reached from the current state. This greatly reduces the complexity of determining a decision.

The second key ingredient of our algorithmic approach was the use of rollout strategies to obtain approximations to the future costs associated with a current decision. Rollout strategies were proposed, by Tesauro [5] and others, for evaluating surrogate future consequences associated with current decisions. The key to rollout strategies is to use a suboptimal strategy to model how future decisions are made as a consequence of current outcomes. Given this suboptimal strategy, future performance can be evaluated either through analytical means or simulation. The expected future performance is combined with the benefits of current actions to select the desired control.

From a theoretical perspective, rollout strategies correspond to a policy improvement step in the policy iteration algorithm for solution of stochastic dynamic programming problems [3]. Starting from the suboptimal strategy, the algorithm computes an improved decision, which takes into account the future consequences. In our work, we investigated the sensitivity of performance to the choice of suboptimal strategy, and established that we recovered close to 90% of the optimal performance from a single policy improvement step using rollout strategies. These results are documented in the paper in Appendix A.

4.3 APPROXIMATE DYNAMIC PROGRAMMING FOR THE SOLUTION OF MULTIPLATFORM PATH PLANNING PROBLEMS

As an extension of our previous work, we consider approximate dynamic programming techniques for the general multi-platform path planning problem described in section 3.1. A specific test problem was selected for which the optimal policy could be computed using dynamic programming. For this problem, we consider sub-optimal policies that are based on the solution of a deterministic auxiliary problem, as well as an exact rollout policy.

The deterministic auxiliary problem is formulated as presented in section 3.1, except that the probabilities associated with each arc are set to one, thus making the state transition dynamics deterministic. This simplifies the original stochastic problem, but the integer programming problem that remains is a vehicle routing problem where the vehicles are constrained to travel along arcs of a graph, which is very difficult to solve computationally. Dynamic programming is used to solve the auxiliary problem in this case, more efficient heuristics are considered in subsequent sections.

Using the solution to the dynamic programming problem, we considered the use of two heuristic policies. The first heuristic policy computes the optimal policy for the deterministic auxiliary problem, starting from a specific state, and implements the actions in the stochastic problem that would be optimal for the same state in the auxiliary problem. The second heuristic policy uses the cost-to-go value computed

for the deterministic auxiliary problem as an approximate cost-to-go in the stochastic problem. In this case, actions are implemented to maximize the sum of the actual incremental cost and an expected cost-to-go.

In addition to the previous two policies, we investigated the use of a "rollout" heuristic in which an arbitrary policy is used to compute the expected cost-to-go for every state under that policy. Although the expected cost-to-go is typically estimated on-line using Monte Carlo simulation of the base policy, we evaluated the cost-to-go exactly using dynamic programming. Such exact evaluations are impractical for larger problems, but we were able to compute it for our example.

The results obtained using the first heuristic policy indicate that naively applying the optimal control action from the deterministic auxiliary problem often does significantly worse than the optimal policy, especially in the risky cases. However, the second heuristic policy, which approximates the optimal cost-to-go using the cost-to-go from the auxiliary problem, can perform significantly worse than the first heuristic policy. Rollout based on the exact evaluation of the heuristic cost recovers some (but not all) of the optimal expected value. These negative results were surprising, in as much as we expected that a representative approximation to the cost-to-go would be sufficient for near-optimal decisionmaking. As it turns out, the deterministic evaluation of a cost-to-go led to poor performance.

In conclusion, the experimental results show that the introduction of random vehicle destruction has a big effect on the qualitative nature of optimal routing solutions. The results of this investigation are documented in the paper in Appendix B.

4.4 Adaptive Multi-platform Scheduling in a Risky Environment

As a result of our prior investigations, we focused our attention on approximate dynamic programming algorithms which use stochastic evaluations of the cost-to-go functions. We focused our research on the use of rollout algorithms, which use a suboptimal strategy to generate future decisions as a function of future states. Rollout algorithms evaluate the cost-to-go for a given state by simulating the future state trajectories using the suboptimal strategy, and using Monte Carlo statistical methods to estimate the cost-to-go from a given state. An alternative approach is to use off-line simulation to learn a nonlinear function estimator (e.g. a neural net) which would generate an estimate of the cost-to go from specific states.

Our investigations focused on the tradeoff between on-line Monte Carlo evaluation versus off-line estimation of the cost-to-go. A larger test problem was considered for this case, for which an optimal policy was not available, so performance was evaluated with respect the baseline suboptimal policy, which was a greedy policy. Monte Carlo simulation was used for both on-line and off-line estimation, and variance reduction was employed to reduce sensitivity to the random events (i.e., platform loss).

The greedy strategy selected the action that maximized the expected value of the current incremental cost without considering the future (i.e., the cost-to-go was approximated by zero). The computational overhead was further reduced by consider the platforms sequentially, and the feasible control set was constrained to account for the finite horizon and the return home requirement.

On-line methods estimated the greedy cost-to-go for a given state and control option by executing the control option, simulating the outcome (i.e., loss of platforms), and than executing and simulating greedy control options throughout the remainder of the time horizon. By averaging over many trajectories, we were able to estimate the cost-to-go of the greedy policy. The control actions are selected such that the sum of the actual incremental cost and the greedy cost-to-go estimate is maximized. In this effort, we considered the sensitivity of the on-line methods to the number of Monte Carlo runs used to estimate the cost-to-go and to the length of the simulation horizon.

The off-line methods trained a parametric approximation of the cost-to-go that is based on features of the current state. The features correspond to the deterministic certainty equivalent problems similar to those used in section 3.2. As an example of the features used, one feature was the value achieved in a deterministic problem where the probability of successfully traversing arcs whose probability is greater than some specified threshold was set to one, while the remaining arcs had probabilities set to zero. Using different threshold values generates a variety of such features. Approximations based on 2 and 4 weighted features were considered. In addition, for the 2-feature approximation, the parametric weightings were optimized.

The results of our on-line experiment show that rollout performed significantly better than the underlying heuristic policies, using only a modest number of Monte Carlo trajectories. We also demonstrated sensitivity to both the length of the horizon and the number of Monte Carlo runs used to estimate the cost-to-go. For the test case considered, good performance was achieved using 20 Monte Carlo simulations and a horizon of no less than 6 out of the 10 steps. The results of our off-line experiments have shown surprisingly, that training the parametric weights seldom approached the performance of the optimized weights. We also observe that rollout algorithms based on parametric approximations and off-line training failed to achieve the level of performance of similar rollout algorithms using on-line Monte Carlo simulations. Exploration of alternative approximations using different features is an area for future investigations. These results are documented in the paper in Appendix C.

4.5 Dynamic Programming Methods for Adaptive Multi-platform Scheduling in a Risky Environment

In the previous section, we observed that using on-line Monte Carlo simulations to evaluate the reference base heuristic policies performed significantly better than the base policies as well as off-line training methods. However, even using a modest number of Monte Carlo simulations resulted in large computation times. Here we consider alternatives to using on-line simulation. In particular, we consider approaches that use analytic approximations of the value function.

We first consider a class of approximation techniques, termed limited-lookahead, in which control actions are implemented at the current state such that the cumulative cost over several stages (the lookahead) plus an approximation of the cost-to-go from the resulting state is maximized. The previously discussed rollout algorithms are a special case in which a single-stage policy is employed and on-line simulation is used in combination with a base heuristic to approximate the cost-to-go. We focus here on limited lookahead policies that solve optimally for a limited horizon while approximating the value of the remaining stages with a simple heuristic. A special case of such policies in which the value-to-go is approximated with zero is referred to in the literature as rolling or receding horizon procedures [10]. Generally, the effectiveness of limited lookahead policies depends on two factors:

1) The quality of the value-to-go approximation—performance of the policy typically improves with approximation quality.

2) The length of the lookahead horizon—performance of a policy typically improves as the horizon becomes longer (at least for small horizon lengths, e.g., 1-4).

However, as the size of the lookahead increases, the number of possible states that can be visited increases exponentially. To keep the overall computation practical, the complexity of the cost-to-go approximation should be reduced for larger lookahead sizes. Balancing such tradeoffs is therefore a critical element in determining the size of the lookahead and the method for approximating the cost-to-go. To reduce computation requirements in this work, we investigated incremental pruning techniques, in which all but the B best one-step lookahead values are pruned. Where trial and error are typically used to determine B, the branching factor.

Our second approach exploits the structure of the problem by decomposing the problem into sub-problems associated with each platform. These sub-problem are solved sequentially, taking into account the solutions of the previous sub-problems. Each sub-problem determines the optimal sequence of nodes to visit assuming that the associated platform was the only one available. The optimal sequence may be computed analytically. In subsequent sub-problems, each node is updated to reflect the probability that that node was not previously visited. This allows each platform to account for previously scheduled platforms. Since the order of the platforms (sub-problems) will impact the overall performance, we considered a fixed ordering according to platform value, enumerating all orderings, and a combinatorial rollout [9] in which the order is determined one vehicle at a time. In addition, we also consider several different ways in which to apply the platform decomposition heuristic to our problem: applied once to obtain a policy for all stages, reapplied at every stage to obtain an updated policy based on new information, and as a heuristic for the limited lookahead policies.

Limited lookahead policies were evaluated for lookahead horizons from 1 to 3, and several simple cost-to-go approximations were considered. These results were relatively insensitive to the cost-to-go approximations, but generated slightly better performance than obtained by rollout algorithms with on-line simulation (section 3.4) and with similar computation requirements. Pruning significantly reduced computation time without loss in performance.

The decomposition approach produced results that were extremely close to the optimal values and required small computation times. The simplest application, in which the heuristic was applied once to obtain a policy for all stages, performed comparably to 2-stage lookahead policies, and the other variations were able to obtain strategies that yielded results that were less than one percent from the optimal expected results. However, this method is limited to problems with suitable problem structure, whereas the limited lookahead methods may easily be generalized to other problems.

These results are documented in the paper in Appendix D.

4.6 Approximate Dynamic Programming for Multi-Vehicle Scheduling in a Risky Environment

In these investigations, we extended our prior model (section 3.1) to include precedence constraints among platform types such that coordination is required to complete the task at specific nodes. Mathematically, we assume that there are two types of platforms in the problem, and that nodes with precedence constraints require a visit from a platform of type 1 before or concurrent with a visit from a platform of type 2 before the task value at the node is collected. Let N_1 denote the nodes with precedence constraints, and let N_2 denote the rest of the nodes. For nodes n in N_1, we modify the definition of node state as follows:

$$v_k(n) = \begin{cases} 2 \text{ if no platform has visited node } n \text{ before or up to stage } k \\ 1 \text{ if a platform of type 1 has visited node } n \text{ before or up to stage } k, \text{ but not one of type 2} \\ 0 \text{ if both platform types visited the node in the right order} \end{cases}$$

The state dynamics for nodes with precedence constraints are straightforward to define. If a platform of type 1 reaches the node at stage k, and its state is $v_{k-1}(n) = 2$, its state switches to $v_k(n) = 1$. If a platform of type 2 reaches the node at stage k, and its state is $v_{k-1}(n) = 2$, there is no state transition; if $v_{k-1}(n) = 1$, then $v_k(n) = 0$. If platforms of both types reach the node at stage k, then $v_k(n) = 0$.

The coupling required for platforms to accomplish a task greatly increased the number of combinations of actions, which must be considered for determining an optimal or near-optimal control. Furthermore, it limited the applicability of platform decomposition approaches. In order to obtain an approximation to the cost-to-go, we formulated the coupled scheduling problem as a bundled assignment problem [11-13], where platform-times are matched with tasks. Specifically, bundles of platform-time combinations are assigned to a given task, and achieve a desired value, depending on the nature of the bundle.

It is well known that such bundle assignment problems are NP-hard. We developed a new approach for the approximate solution of these bundle assignment problems, based on the use of combinatorial rollout ideas [9] and approximations using standard assignment problems. The results show improved performance with very small computation cost. These results are also documented in the paper in Appendix D.

5 Technology Transition and Future Research

ALPHATECH has transition the technology developed under this contract to DARPA's Agile Control of Military Operations (JFACC) program. In this program, ALPHATECH's technology development is focused on providing the military commander with real-time, optimal control of military air operations via near-optimal mission assignments which anticipate possible mission modifications due to uncertain future events over a 24-hour segment of a JAO campaign. The primary benefit of this technology is agile and stable control of distributed and dynamic military operations conducted in inherently uncertain, hostile, and rapidly changing environments. Key features of the JAO environment of interest include risk and reward that are dependent on package composition and weaponeering.

Like the problem considered under this contract, the JFACC problem includes combinatorial, stochastic, and temporal complexity that must be efficiently managed; however, the JFACC program has the additional complexity of heterogeneous assets which must make multiple turns in order to achieve mission objectives. Accordingly, much of the technology developed under this contract was transferred to the JFACC program. The results to date have demonstrated the benefits of feedback control and near-optimal control for the JAO problem. Through experimentation, it was shown that feedback control was able to desensitize the controller performance to environmental uncertainty. In other word, feedback control provides agility. Additionally, the optimal control framework demonstrates the ability to produce operationally consistent behavior by anticipating key uncertainties and positioning assets for opportunities of recourse.

Given these promising results obtained from the JFACC program and those from New World Vistas, ALPHATECH plans on expanding this work to DARPA/ITO's Man and Machine Command and Control (M2C2) program. The M2C2 program is currently under development and is expected as a follow-on to the current JFACC program. The focus of the M2C2 program is to provide agile, autonomous control of a tactical air and ground campaign. This problem has similar complexities to that of the JFACC problem with the additional complexity that the sensors and shooters are both ground and air based.

ALPHATECH is also actively marketing the research developed under this AFOSR contract to solve the problem of distributed, cooperative control for a team of autonomous, tactical UAVs. To realize the potential of autonomous assets in a tactical operational setting, the complex problem of cooperative, autonomous control must be solved. Since multiple assets must operate in a common battlespace, cooperative control is required to maximize operational efficiency. In cooperative control problems, distributed assets develop and pursue a common strategy, and each vehicle conducts its part of the strategy through local control actions. Thus, decisions that are currently made by human operators—tactical routing, coordinated attack, store release, etc.—will need to be generated by an intelligent cooperative control system. This cooperative control system will have to solve the large-scale, complex dynamic optimization problems associated with mission planning and control, in an unstructured and uncertain environment, in near real time, and without human intervention. ALPHATECH intends on transitioning many of the techniques developed under this contract to this problem.

6 SUMMARY

Our objective at the beginning of our research was the development of a new paradigm and the required algorithms for real-time planning and scheduling in the presence of uncertainties. This paradigm would not simply replan in the presence of contingencies, but would anticipate the possible occurrence of these contingencies, and would therefore generate robust plans that can readily accommodate contingencies whenever they occurred.

The core algorithmic idea in our approach was the use of approximate stochastic dynamic programming (SDP). In contrast with standard SDP approaches, which considered every possible future state a priori, and developed an optimal decision for each state, we proposed a real-time control strategy, which was based on just-in-time computation of control actions. That is, the control for the current state was not determined by an a priori off-line strategy, but rather by an on-line optimization problem, in the manner of model-predictive control [14]. The advantage of this just-in-time strategy is that the only states which needed to be considered were the states which actually happened in the problem realization, plus nearby states which were likely to be reached from actual states.

The key issues that we needed to investigate were alternative approaches at defining this on-line optimization problem to trade off computation complexity and near-optimal performance. The papers in the appendices document the different investigations. The major lessons learned were:

1. The use of neural networks and other nonlinear approximations for estimating future costs, when trained using simulation data led to unsatisfactory performance for stochastic scheduling. The basic reason was that, when performing off-line training, one needs to train the network over a range of possible problem instances. The approximations investigated had difficulty generating accurate estimates across different problem instances.

2. Using a surrogate control policy, together with real-time Monte Carlo evaluation of the performance of this policy in a rollout approach provided near-optimal performance. However, the Monte Carlo evaluation required extensive computation, making it ill-suited for the just-in-time optimization paradigm.
3. One of the key problems not addressed by approximate SDP is the required enumeration of combinatorially large numbers of possible control decisions at a specific time. A key requirement for computation efficiency is to develop techniques that accelerate this combinatorial optimization, particularly for problems with large numbers of platforms.
4. The approach that worked best was based on using decompositions and local optimization problems, which were matched to the problem structure, but not the problem instance.
5. The strategy of approximate dynamic programming can yield near-optimal performance at a fraction of the computation complexity, but requires careful selection of the approximation technique.

Some important directions for future investigation include addressing problems with partial observations of the state, and developing approaches that scale to problems with larger numbers of platforms.

REFERENCES

1. Chairman of the Joint Chiefs of Staff, "Joint Vision 2010," 5126 Joint Staff, Pentagon, Washington, D.C. 20318-5126, 1999.
2. USAF Scientific Advisory Board, "New World Vistas: Air and Space Power for the 21st Century," AF/SB, December 1995.
3. Bertsekas, D.P., *Dynamic Programming and Optimal Control, Vol. I*, 2nd Edition, Athena Scientific, 2000.
4. Bertsekas, D.P., and Tsitsiklis, J.N., *Neuro-Dynamic Programming*, Athena Scientific, Belmont, MA, 1996.
5. Tesauro, G., and Galperin, G.R., "On-Line Policy Improvement Using Monte Carlo Search," presented at the 1996 Neural Information Processing Systems Conference, Denver, CO, 1996.
6. Barto, A.G., Bradtke, S.J., and Singh, S.P., "Learning to Act Using Real-Time Dynamic Programming," *Artificial Intelligence*, vol. 72, 1995, pp. 81-138.
7. Bertsekas, D.P., Castañon, D.A., "Rollout Algorithms for Stochastic Scheduling Problems," *Journal of Heuristics*, vol. 5, 1999.
8. Bertsekas, D.P., "A Note on the Robust Calculation of Rollout Policies," Lab. for Information and Decision Systems Report LIDS-P-2392, Mass. Institute of Technology, 1997.
9. Bersekas, D.P., Tsitsiklis, J.N., Wu, C., "Rollout Algorithms for Combinatorial Optimization," *Journal of Heuristics*, vol. 3, 1997, pp. 245-262.
10. Alden, J.M., Smith, R.L., "Rolling Horizon Procedures in Nonhomogeneous Markov Decision Processes," *Operations Research*, vol. 40, 1992.
11. Parkes, D.C., Ungar, L.H., "Iterative Combinatorial Auctions: Theory and Practice," *Proc. 17th National Conference on Artificial Intelligence*, 2000.
12. Rothkopf, M.H., Pekec, A., Harstad, R.M., "Computationally Manageable Combinatorial Auctions," *Management Science*, vol. 44, 1998.
13. Sandholm, T., "An Algorithm for Optimal Winner Determination in Combinatorial Auctions," *Proc. 11th National Conference on Artificial Intelligence*, 1993.
14. Mayne, D.Q, Rawlings, J.B., Rao, C.V., and Scokaert P.O.M., "Constrained Model Predictive Control: Stability and Optimality," Automatica, Vol. 36, 2000, pp. 789-814.

APPENDICES AND ATTACHMENTS

Bertsekas, D.P., Castañon, D.A., "Rollout Algorithms for Stochastic Scheduling Problems," *Journal of Heuristics*, vol. 5, 1999.

Patek, S.D., Logan, D.A., Castañon, D.A., "Approximate Dynamic Programming for the Solution of Multiplatform Path Planning Problems," *Proc. 1999 IEEE International Conference on Systems, Man, and Cybernetics*, 1999.

Bertsekas, D.P., Castañon, D.A., Curry, M.L., Logan, D.A., "Adaptive Multi-platform Scheduling in a Risky Environment," *1999 Proceedings from Advances in Enterprise Control Symposium*, 1999.

Bertsekas, D.P., Castañon, D.A., Curry, M.L., Logan, D.A., Cynara, W., "Dynamic Programming Methods for Adaptive Multi-platform Scheduling in a Risky Environment," *2000 Proceedings from Advances in Enterprise Control Symposium*, 2000.

Cynara, W., Bertsekas, D.P., Castañon, D.A., Curry, M.L., Logan, D.A., "Approximate Dynamic Programming for Multi-vehicle Scheduling in a Risky Environment," to be submitted to the IEEE Transactions on Systems, Man, and Cybernetics.

ROLLOUT ALGORITHMS FOR STOCHASTIC SCHEDULING PROBLEMS [1]

Dimitri P. Bertsekas
Dept. of Electrical Engineering and Computer Science
M. I. T.
Cambridge, Mass., 02139

David A. Castañon
Dept. of Electrical and Computer Engineering
Boston University
Boston, MA 02215

Abstract

Stochastic scheduling problems are difficult stochastic control problems with combinatorial decision spaces. In this paper we focus on a class of stochastic scheduling problems, the quiz problem and its variations. We discuss the use of heuristics for their solution, and we propose rollout algorithms based on these heuristics which approximate the stochastic dynamic programming algorithm. We show how the rollout algorithms can be implemented efficiently, and we delineate circumstances under which they are guaranteed to perform better than the heuristics on which they are based. We also show computational results which suggest that the performance of the rollout policies is near-optimal, and is substantially better than the performance of their underlying heuristics.

Journal of Heuristics, V. 5, pp. 93–112, (1999).

[1] This work was supported in part by the Air Force Office of Scientific Research under grant no. F49620-97-C-0013 and the Air Force Research Laboratory under grant F33615-96-C-1930

1. INTRODUCTION

Consider the following variation of a planning problem: There is a finite set of locations which contain tasks of interest, of differing value. There is a single processor on which the tasks are to be scheduled. Associated with each task is a task-dependent risk that, while executing that task, the processor will be damaged and no further tasks will be processed. The objective is to find the optimal task schedule in order to maximize the expected value of the completed tasks.

The above is an example of a class of stochastic scheduling problems known in the literature as *quiz problems* (see Bertsekas [1995], Ross [1983], or Whittle [1982]). The simplest form of this problem involves a quiz contest where a person is given a list of N questions and can answer these questions in any order he or she chooses. Question i will be answered correctly with probability p_i, and the person will then receive a reward v_i. At the first incorrect answer, the quiz terminates and the person is allowed to keep his or her previous rewards. The problem is to choose the ordering of questions so as to maximize expected rewards.

The problem can be viewed in terms of dynamic programming (DP for short), but can more simply be viewed as a deterministic combinatorial problem, whereby we are seeking an optimal sequence in which to answer the questions. It is well-known that the optimal sequence is deterministic, and can be obtained using an interchange argument; questions should be answered in decreasing order of $p_i v_i/(1 \quad p_i)$. This will be referred to as the *index policy*. An answer order is optimal if and only if it corresponds to an index policy. Another interesting simple policy for the quiz problem is the *greedy policy*, which answers questions in decreasing order of their expected reward $p_i v_i$. A greedy policy is suboptimal, essentially because it does not consider the future opportunity loss resulting from an incorrect answer.

Unfortunately, with only minor changes in the structure of the problem, the optimal solution becomes much more complicated (although DP and interchange arguments are still relevant). Examples of interesting and difficult variations of the problem involve one or more of the following characteristics:

(a) A limit on the maximum number of questions that can be answered, which is smaller than the number of questions N. To see that the index policy is not optimal anymore, consider the case where there are two questions, only one of which may be answered. Then it is optimal to use the greedy policy rather than the index policy.

(b) A time window for each question, which constrains the set of time slots when each question may be answered. Time windows may also be combined with the option to refuse answering

a question at a given period, when either no question is available during the period, or answering any one of the available questions involves excessive risk.

(c) Precedence constraints, whereby the set of questions that can be answered in a given time slot depends on the immediately preceding question, and possibly on some earlier answered questions.

(d) Sequence-dependent rewards, whereby the reward from answering correctly a given question depends on the immediately preceding question, and possibly on some questions answered earlier.

It is clear that the quiz problem variants listed above encompass a very large collection of practical scheduling problems. The version of the problem with time windows and precedence constraints relates to vehicle routing problems (involving a single vehicle). The version of the problem with sequence-dependent rewards, and a number of questions that is equal to the maximum number of answers relates to the traveling salesman problem. Thus, in general, it is very difficult to solve the variants described above exactly.

An important feature of the quiz problem, which is absent in the classical versions of vehicle routing and traveling salesman problems is that *there is a random mechanism for termination of the quiz*. Despite the randomness in the problem, however, in all of the preceding variants, there is an *optimal open-loop policy*, i.e., an optimal order for the questions that does not depend on the random outcome of the earlier questions. The reason is that we do not need to plan the answer sequence following the event of an incorrect answer, because the quiz terminates when this event occurs. Thus, we refer to the above variations of the quiz problem as *deterministic quiz problems*.

There are variants of the quiz problem where the optimal order to answer questions depends on random events. Examples of these are:

(e) There is a random mechanism by which the quiz taker may miss a turn, i.e., be denied the opportunity to answer a question at a given period, but may continue answering questions at future time periods.

(f) New questions can appear and/or old questions can disappear in the course of the quiz according to some random mechanism. A similar case arises when the start and end of the time windows can change randomly during the quiz.

(g) There may be multiple quiz takers that answer questions individually, and drop out of the quiz upon their own first error, while the remaining quiz takers continue to answer questions.

(h) The quiz taker may be allowed multiple chances, i.e., may continue answering questions up to a given number of errors.

(i) The reward for answering a given question may be random and may be revealed to the quiz taker at various points during the course of the quiz.

The variants (e)-(i) of the quiz problem described above require a genuinely stochastic formulation as Markovian decision problems. We refer to these variations in the paper as *stochastic quiz problems*. They can be solved exactly only with DP, but their optimal solution is prohibitively difficult. This is because the states over which DP must be executed are subsets of questions, and the number of these subsets increases exponentially with the number of questions.

In this paper, we develop suboptimal solution approaches that are computationally tractable for both deterministic and stochastic quiz problems. In particular, we focus on rollout algorithms, a class of suboptimal solution methods inspired from the policy iteration methodology of DP and the approximate policy iteration methodology of neuro-dynamic programming (NDP for short). One may view a rollout algorithm as a single step of the classical policy iteration method, starting from some given easily implementable policy. Algorithms of this type have been sporadically suggested in several DP application contexts. They have also been proposed by Tesauro [1996] in the context of simulation-based computer backgammon (the name "rollout" was introduced by Tesauro as a synonym for repeatedly playing out a given backgammon position to calculate by Monte Carlo averaging the expected game score starting from that position).

Rollout algorithms were first proposed for the approximate solution of discrete optimization problems by Bertsekas and Tsitsiklis [1996], and by Bertsekas, Tsitsiklis, and Wu [1997], and the methodology developed here for the quiz problem strongly relates to the ideas in these sources. Generally, rollout algorithms are capable of magnifying the effectiveness of any given heuristic algorithm through sequential application. This is due to the policy improvement mechanism of the underlying policy iteration process.

In the next section, we introduce rollout algorithms for deterministic quiz problems, where the optimal order for the questions from a given period onward does not depend on earlier random events. In Section 3, we provide computational results indicating that rollout algorithms can improve impressively on the performance of their underlying heuristics. In Sections 4 and 5, we extend the rollout methodology to stochastic quiz problems [cf. variants (e)-(i) above], that require the use of stochastic DP for their optimal solution. Here we introduce the idea of *multiple scenaria* for the future uncertainty starting from a given state, and we show how these scenaria can be used to construct an approximation to the optimal value function of the

problem using NDP techniques and a process of *scenario aggregation*. In Section 6, we provide computational results using rollout algorithms for stochastic quiz problems. Finally, in Section 7, we provide computational results using rollout algorithms for quiz problems that involve graph-based precedence constraints. Our results indicate consistent and substantial .

2. ROLLOUT ALGORITHMS FOR DETERMINISTIC QUIZ PROBLEMS

Consider a variation of a quiz problem of the type described in (a)-(c) above. Let N denote the number of questions available, and let M denote the maximum number of questions which may be attempted. Associated with each question i is a value v_i. and a probability of successfully answering that question p_i. Assume that there are constraints such as time windows or precedence constraints which restrict the possible question orders. Denote by $V(i_1,\ldots,i_M)$ the expected reward of a feasible question order $(i_1,\ldots,i_M)$:

$$V(i_1,\ldots,i_M) = p_{i_1}\ v_{i_1} + p_{i_2}\ v_{i_2} + p_{i_3}(\cdots + p_{i_M} v_{i_M})\cdots)). \tag{2.1}$$

For an infeasible question order $(i_1,\ldots,i_M)$, we use the convention

$$V(i_1,\ldots,i_M) = \infty \quad .$$

The classical quiz problem is the case where $M = N$, and all question orders are feasible. In this case, the optimal solution is simply obtained by using an interchange argument. Let i and j be the kth and $(k+1)$st questions in an optimally ordered list

$$L = (i_1,\ldots,i_{k-1},i,j,i_{k+2},\ldots,i_N).$$

Consider the list

$$L' = (i_1,\ldots,i_{k-1},j,i,i_{k+2},\ldots,i_N)$$

obtained from L by interchanging the order of questions i and j. We compare the expected rewards of L and L'. We have

$$\begin{aligned} E\{\text{reward of } L\} &= E\{\text{reward of } \{i_1,\ldots,i_{k-1}\}\} \\ &\quad + p_{i_1}\cdots p_{i_{k-1}}(p_i v_i + p_i p_j v_j) \\ &\quad + p_{i_1}\cdots p_{i_{k-1}} p_i p_j E\{\text{reward of } \{i_{k+2},\ldots,i_N\}\} \\ E\{\text{reward of } L'\} &= E\{\text{reward of } \{i_1,\ldots,i_{k-1}\}\} \\ &\quad + p_{i_1}\cdots p_{i_{k-1}}(p_j v_j + p_j p_i v_i) \\ &\quad + p_{i_1}\cdots p_{i_{k-1}} p_j p_i E\{\text{reward of } \{i_{k+2},\ldots,i_N\}\}. \end{aligned}$$

Since L is optimally ordered, we have

$$E\{\text{reward of } L\} \geq E\{\text{reward of } L'\},$$

so it follows from these equations that

$$p_i v_i + p_i p_j v_j \geq p_j v_j + p_j p_i v_i$$

or equivalently

$$\frac{p_i v_i}{1 \quad p_i} \geq \frac{p_j v_j}{1 \quad p_j}.$$

It follows that to maximize expected rewards, questions should be answered in decreasing order of $p_i v_i/(1 \quad p_i)$, which yields the index policy.

Unfortunately, the above argument breaks down when either $M < N$, or there are constraints on the admissibility of sequences due to time windows, sequence-dependent constraints, or precedence constraints. For these cases, we can still use heuristics such as the index policy or the greedy policy, but they will not provide optimal performance.

Consider a heuristic algorithm, which given a *partial schedule* $P = (i_1, \ldots, i_k)$ of distinct questions constructs a *complementary schedule* $\overline{P} = (i_{k+1}, \ldots, i_M)$ of distinct questions such that $P \cap \overline{P} = \emptyset$. The heuristic algorithm is referred to as the *base heuristic*. We define the *heuristic reward* of the partial schedule P as

$$H(P) = V(i_1, \ldots, i_k, i_{k+1} \ldots, i_M). \tag{2.2}$$

If $P = (i_1, \ldots, i_M)$ is a complete solution, by convention the heuristic reward of P is the true expected reward $V(i_1, \ldots, i_M)$.

Given a base heuristic, the corresponding *rollout algorithm* constructs a complete schedule in M stages, one question per stage. The rollout algorithm can be described as follows:

At the 1st stage it selects question i_1 according to

$$i_1 = \arg \max_{i=1,\ldots,N} H(i), \tag{2.3}$$

and at the kth stage ($k > 1$) it selects i_k according to

$$i_k = \arg \max_{\{i \mid i \neq i_1, \ldots, i_{k \quad 1}\}} H(i_1, \ldots, i_{k \quad 1}, i), \qquad k = 2, \ldots, M. \tag{2.4}$$

Thus a rollout policy involves $N + (N \quad 1) + \cdots + (N \quad M) = O(MN)$ applications of the base heuristic and corresponding calculations of expected reward of the form (2.1). While this

is a significant increase over the calculations required to apply the base heuristic and compute its expected reward, the rollout policy is still computationally tractable. In particular, if the running time of the base heuristic is polynomial, so is the running time of the corresponding rollout algorithm. On the other hand, it will be shown shortly that the expected reward of the rollout policy is at least as large as the one of the base heuristic.

As an example of a rollout algorithm, consider the special variant (a) of the quiz problem in the preceding section, where at most M out of N questions may be answered and there are no time windows or other complications. Let us use as base heuristic the index heuristic, which given a partial schedule $(i_1, \ldots, i_k)$, attempts the remaining questions according to the index policy, in decreasing order of $p_i v_i/(1 \quad p_i)$. The calculation of $H(i_1, \ldots, i_k)$ is done using Eq. (2.1), once the questions have been sorted in decreasing order of index. The corresponding rollout algorithm, given $(i_1, \ldots, i_{k \ 1})$ selects i, calculates $H(i_1, \ldots, i_{k \ 1}, i)$ for all $i \neq i_1, \ldots, i_{k \ 1}$, using Eq. (2.1), and then optimizes this expression over i to select i_k.

Note that one may use a different heuristic, such as the greedy heuristic, in place of the index heuristic. There are also other possibilities for base heuristics. For example, one may first construct a complementary schedule using the index heuristic, and then try to improve this schedule by using a 2-OPT local search heuristic, that involves exchanges of positions of pairs of questions. One may also use multiple heuristics, which produce heuristic values $H_j(i_1, \ldots, i_k)$, $j = 1, \ldots, J$, of a generic partial schedule $(i_1, \ldots, i_k)$, and then combine them into a "superheuristic" that gives the maximal value

$$H(i_1, \ldots, i_k) = \max_{j=1,\ldots,J} H_j(i_1, \ldots, i_k).$$

An important question is whether the rollout algorithm performs at least as well as its base heuristic when started from the initial partial schedule. This can be guaranteed if the base heuristic is *sequentially consistent*. By this we mean that the heuristic has the following property:

Suppose that starting from a partial schedule

$$P = (i_1, \ldots, i_{k \ 1}).$$

the heuristic produces the complementary schedule

$$\overline{P} = (i_k, \ldots, i_M).$$

Then starting from the partial schedule

$$P^+ = (i_1, \ldots, i_{k \ 1}, i_k),$$

the heuristic produces the complementary schedule

$$\overline{P}^{+} = (i_{k+1}, \ldots, i_M).$$

As an example, it can be seen that the index and the greedy heuristics, discussed earlier, are sequentially consistent. This is a manifestation of a more general property: many common base heuristics of the greedy type are by nature sequentially consistent. It may be verified, based on Eq. (2.4), that a sequentially consistent rollout algorithm keeps generating the same schedule $P \cup \overline{P}$, up to the point where by examining the alternatives in Eq. (2.4) and by calculating their heuristic rewards, it discovers a better schedule. As a result, sequential consistency guarantees that the reward of the schedules $P \cup \overline{P}$ produced by the rollout algorithm is monotonically nonincreasing: that is, we have

$$H(P^{+}) \leq H(P)$$

at every stage. For further elaboration of the sequential consistency property, we refer to the paper by Bertsekas, Tsitsiklis, and Wu [1997], which also discusses some underlying connections with the policy iteration method of dynamic programming.

A condition that is more general than sequential consistency is that the algorithm be *sequentially improving*, in the sense that at each stage there holds

$$H(P^{+}) \leq H(P).$$

This property also guarantees that the rewards of the schedules produced by the rollout algorithm are monotonically nonincreasing. The paper by Bertsekas, Tsitsiklis, and Wu [1997] discusses situations where this property holds, and shows that with fairly simple modification, a rollout algorithm can be made sequentially improving.

There are a number of variations of the basic rollout algorithm described above. In particular, we may incorporate *multistep lookahead* or *selective depth lookahead* into the rollout framework. An example of a rollout algorithm with m-step lookahead operates as follows: at the kth stage we augment the current partial schedule $P = (i_1, \ldots, i_{k-1})$ with all possible sequences of m questions $i \neq i_1, \ldots, i_{k-1}$. We run the base heuristic from each of the corresponding augmented partial schedules, we select the m-question sequence with maximum heuristic reward, and then augment the current partial schedule P with the first question in this sequence. An example of a rollout algorithm with *selective* two-step lookahead operates as follows: at the kth stage we start with the current partial schedule $P = (i_1, \ldots, i_{k-1})$, and we run the base heuristic starting from each partial schedule $(i_1, \ldots, i_{k-1}, i)$ with $i \neq i_1, \ldots, i_{k-1}$. We then form the subset $\overline{I}$ consisting of the n questions $i \neq i_1, \ldots, i_k$ that correspond to the n best complete schedules thus

obtained. We run the base heuristic starting from each of the partial schedules $(i_1, \ldots, i_{k-1}, i, j)$ with $i \in \bar{I}$ and $j \neq i_1, \ldots, i_{k-1}, i$, and obtain a corresponding complete schedule. We then select as next question i_k of the rollout schedule the question $i \in \bar{I}$ that corresponds to a maximal reward schedule. Note that by choosing the number n to be smaller than the maximum possible, $N - k+1$, we can reduce substantially the computational requirements of the two-step lookahead.

3. COMPUTATIONAL EXPERIMENTS WITH DETERMINISTIC QUIZ PROBLEMS

In order to explore the performance of rollout algorithms for deterministic scheduling, we conducted a series of computational experiments involving the following seven algorithms:

(1) The optimal stochastic dynamic programming algorithm.

(2) The greedy heuristic, where questions are ranked in order of decreasing $p_i v_i$, and, for each stage k, the feasible unanswered question with the highest ranking is selected.

(3) The index heuristic, where questions are ranked in order of decreasing $p_i v_i/(1 - p_i v_i)$, and for each stage k, the feasible unanswered question with the highest ranking is selected.

(4) The one-step rollout policy based on the greedy heuristic, where, at each stage k, for every feasible unanswered question i_k and prior sequence $i_1, \ldots, i_{k-1}$, the question is chosen according to the rollout rule (2.4), where the function H uses the greedy heuristic as the base policy.

(5) The one-step rollout policy based on the index heuristic, where the function H in (2.4) uses the index heuristic as the base policy,

(6) The selective two-step lookahead rollout policy based on the greedy heuristic. At the k-th stage, the base heuristic is used in a one-step rollout to select the best four choices for the current question among the admissible choices. For each of these choices at stage k, the feasible continuations at stage $k+1$ are evaluated using the greedy heuristic to complete the schedule. The choice at stage k is then selected from the sequence with the highest evaluation.

(7) The selective two-step lookahead rollout policy based on the index heuristic.

The problems selected for evaluation involve 20 possible questions and 20 stages, which are small enough so that exact solution using dynamic programming is possible. Associated with each

question is a sequence of times, determined randomly for each experiment, when that question can be attempted. Floating point values were assigned randomly to each question from 1 to 10 in each problem instance. The probabilities of successfully answering each question were also chosen randomly, between a specified lower bound and 1.0. In order to evaluate the performance of the last six algorithms, each suboptimal algorithm was simulated 10,000 times, using independent event sequences determining which questions were answered correctly.

Our experiments focused on the effects of two factors on the relative performance of the different algorithms:

(a) The lower bound on the probability of successfully answering a question, which varied from 0.2 to 0.8

(b) The average percent of questions that are admissible (i.e., that can be answered) at any one stage, which ranged from 10% to 50%.

The first set of experiments fixed the average percentage of questions which can be answered at a single stage to 10%, and varied the lower bound on the probability of successfully answering a question across four conditions: 0.2, 0.4, 0.6 and 0.8. For each experimental condition, we generated 30 independent problems and solved them, and evaluated the corresponding performance using 10,000 Monte Carlo runs. We computed the average performance across the 30 problems, and compared this performance with the performance obtained using the stochastic dynamic programming algorithm.

Table 1 shows the results of our experiments. The average performance of the greedy and index heuristics in each condition are expressed in terms of the percentage of the optimal performance. For low probability of success, both heuristics obtain less than half of the performance of the optimal algorithm. The table also illustrates the improvement in performance obtained by both the one-step rollout and the selective two-step rollout algorithms, expressed in terms of percentage of the optimal performance. As an example, the first column of Table 1 gives the average performance across 30 problems with lower bound on the probability of successfully answering a question 0.2. The performance achieved by the greedy heuristic was 41% of optimal, whereas the average performance of the one-step rollout with the greedy heuristic as a base policy achieved on average 75% of the optimal performance, which was a 34% improvement. Furthermore, the two-step selective rollout achieved on average 81% of the optimal performance.

The results in Table 1 show that one-step rollouts significantly improve the performance of both the greedy and the index heuristics in these difficult stochastic combinatorial problems. In particular, the rollout algorithms recovered in all cases at least 50% of the loss of value due to

Minimum Probability of Success	0.2	0.4	0.6	0.8
Greedy Heuristic	41%	50%	61%	76%
Improvement by One-step Rollout	34%	32%	27%	14%
Improvement by Two-step Rollout	40%	34%	27%	14%
Index Heuristic	43%	53%	66%	80%
Improvement by One-step Rollout	34%	30%	23%	10%
Improvement by Two-step Rollout	38%	33%	24%	11%

Table 1: Performance of the different algorithms as the minimum probability of success of answering a question varies. The average percentage of questions which can be answered at a single stage is fixed at 10%. The numbers reported are percentage of the performance of the optimal dynamic programming solution achieved, averaged across 30 independent problems. As an example, the first column gives the average performance across 30 problems with lower bound on the probability of successfully answering a question 0.2. The performance achieved by the greedy heuristic was 41% of optimal, whereas the average performance of the one-step rollout with the greedy heuristic as a base policy achieved on average 75% of the optimal performance, which was a 34% improvement.

the use of the heuristic. Loss recovery of this order or better was typical in all of the experiments with rollout algorithms in this paper. The results also illustrate that the performance of the simple heuristics improves as the average probability of success increases, thereby reducing the potential advantage of rollout strategies. Even in these unfavorable cases, the rollout strategies improved performance levels by at least 10% of the optimal policy, and recovered a substantial portion of the loss due to the suboptimality of the heuristic.

For the size of problems tested in these experiments, the advantages of using a two-step selective lookahead rollout were small. In many cases, the performances of the one-step rollout and the two-step selective lookahead rollout were identical. Nevertheless, for selected difficult individual problems, the two-step selective lookahead rollout improved performance by as much as 40% of the optimal strategy over the level achieved by the one-step rollout with the same base heuristic.

The second set of experiments fixed the lower bound on the probability of successfully

answering a question to 0.2, and varied the average percent of admissible questions at any one stage across 3 levels: 10%, 30% and 50%. As before, we generated 30 independent problems and evaluated the performance of each algorithm on each problem instance. The results of these experiments are summarized in Table 2. As before, the performance of the greedy and index heuristics improves as the experimental condition approaches the standard conditions of the quiz problem, where 100% of the questions can be answered at any time. The results confirm the trend seen in Table 1: even in cases where the heuristics achieve good performance, rollout strategies offer significant performance gains.

Problem Density	0.1	0.3	0.5
Greedy Heuristic	41%	58%	76%
Improvement by One-step Rollout	34%	28%	15%
Improvement by Two-step Rollout	40%	32%	16%
Index Heuristic	43%	68%	85%
Improvement by One-step Rollout	34%	22%	8%
Improvement by Two-step Rollout	38%	24%	9%

Table 2: Performance of the different algorithms as the average number of questions per period increases. The lower bound on the probability of successfully answering a question is fixed at 0.2. The numbers reported are percentage of the performance of the optimal dynamic programming solution achieved, averaged across 30 independent problems.

The results in Tables 1 and 2 suggest that the advantage of rollout strategies over the greedy and index heuristics increases with the risk involved in the problem. This advantage stems from the forward-looking character of rollout strategies. In particular, by constructing a feasible strategy for the entire horizon for evaluating the current decision, rollout strategies account for the limited future accessibility of questions, and compute tradeoffs between future accessibility and the risk of the current choice. In contrast, myopic strategies such as the greedy and index heuristics do not account for future access to questions, and thus are forced to make risky choices when no other alternatives are present. Thus, as the risk of missing a question

increases and the average accessibility of questions decreases, rollout strategies achieve nearly double the performance of the corresponding myopic heuristics.

4. ROLLOUT ALGORITHMS FOR STOCHASTIC QUIZ PROBLEMS

We now consider variants of the quiz problem where there is no optimal policy that is open-loop. The situations (e)-(i) given in Section 1 provide examples of quiz problems of this type. We can view such problems as stochastic DP problems. Their exact solution, however, is prohibitively expensive.

Let us state a quiz problem in the basic form of a dynamic programming problem (see e.g., Bertsekas [1995]), where we have the stationary discrete-time dynamic system

$$x_{k+1} = f_k(x_k, u_k, w_k), \qquad k = 0, 1, \ldots, T. \tag{4.1}$$

that evolves over T time periods. Here x_k is the state taking values in some set, u_k is the control to be selected from a finite set $U_k(x_k)$, w_k is a random disturbance, and f_k is a given function. We assume that the disturbance w_k, $k = 0, 1, \ldots$, has a given probability distribution that depends explicitly only on the current state and control. The one-stage cost function is denoted by $g_k(x, u, w)$. In this general framework, we assume that costs are minimized, but the following discussion can be easily adapted to the case where rewards are maximized.

To apply the rollout framework, we need to have a base policy for making a decision at each state-time pair (x_k, k). We view this policy as a sequence of feedback functions $\pi = \{\mu_0, \mu_1, \ldots, \mu_T\}$, which at time k maps a state x_k to a control $\mu_k(x_k) \in U_k(x_k)$. The cost-to-go of π starting from a state-time pair (x_k, k) will be denoted by

$$J_k(x_k) = E\left\{\sum_{i=k}^{T-1} g_i\big(x_i, \mu_i(x_i), w_i\big)\right\}. \tag{4.2}$$

The cost-to-go functions J_k satisfy the following recursion of dynamic programming (DP for short)

$$J_k(x) = E\big\{g\big(x, \mu_k(x), w\big) + J_{k+1}\big(f\big(x, \mu_k(x), w\big)\big)\big\}, \qquad k = 0, 1, \ldots \tag{4.3}$$

with the initial condition

$$J_T(x) = 0.$$

The *rollout policy based on* π is denoted by $\overline{\pi} = \{\overline{\mu}_0, \overline{\mu}_1, \ldots\}$, and is defined through the operation

$$\overline{\mu}_k(x) = \arg\min_{u \in U(x)} E\big\{g(x, u, w) + J_{k+1}\big(f(x, u, w)\big)\big\}, \qquad \forall\, x,\ k = 0, 1, \ldots. \tag{4.4}$$

Thus the rollout policy is a one step-lookahead policy, with the optimal cost-to-go approximated by the cost-to-go of the base policy. This amounts essentially to a single step of the method of policy iteration. Indeed using standard policy iteration arguments, one can show that the rollout policy $\overline{\pi}$ is an improved policy over the base policy π.

In practice, one typically has a method or algorithm to compute the control $\mu_k(x)$ of the base policy, given the state x, but the corresponding cost-to-go functions J_k may not be known in closed form. Then the exact or approximate computation of the rollout control $\overline{\mu}_k(x)$ using Eq. (4.4) becomes an important and nontrivial issue, since we need for all $u \in U(x)$ the value of

$$Q_k(x,u) = E\big\{g(x,u,w) + J_{k+1}\ f(x,u,w))\big\}. \tag{4.5}$$

known as the *Q-factor* at time k. Alternatively, for the computation of $\overline{\mu}_k(x)$ we need the value of the cost-to-go

$$J_{k+1}\ f(x,u,w))$$

at all possible next states $f(x,u,w)$.

In favorable cases, it is possible to compute the cost-to-go $J_k(x)$ of the base policy π for any time k and state x. An example is the variant of the quiz problem discussed in Sections 2 and 3, where the base policy is an open-loop policy that consists of the schedule generated by the index policy or the greedy policy. The corresponding cost-to-go can then be computed using Eq. (2.1). In general, however, the computation of the cost-to-go of the base policy may be much more difficult. In particular, when the number of states is very large, the DP recursion (4.3) may be infeasible.

A conceptually straightforward approach for computing the rollout control at a given state x and time k is to use Monte Carlo simulation. This was proposed by Tesauro [TeG96] in the context of backgammon. In particular, for a given backgammon position and a given roll of the dice, Tesauro suggested looking at all possible ways to play the given roll, and do a Monte-Carlo evaluation of the expected score starting from the resulting position and using some base computer program to play out the game (for both sides). To implement this approach in the context of a general DP problem, we consider all possible controls $u \in U(x)$ and we generate a "large" number of simulated trajectories of the system starting from x, using u as the first control, and using the policy π thereafter. Thus a simulated trajectory has the form

$$x_{i+1} = f\ x_i, \mu_i(x_i), w_i). \qquad i = k+1, \ldots, T \quad 1,$$

where the first generated state is

$$x_{k+1} = f(x,u,w_k),$$

and each of the disturbances $w_k, \ldots, w_{T-1}$ is an independent random sample from the given distribution. The costs corresponding to these trajectories are averaged to compute an approximation $\tilde{Q}_k(x,u)$ to the Q-factor $Q_k(x,u)$ of Eq. (4.5). The approximation becomes increasingly accurate as the number of simulated trajectories increases. Once the approximate Q-factor $\tilde{Q}_k(x,u)$ corresponding to each control $u \in U(x)$ is computed, we can obtain the (approximate) rollout control $\tilde{\mu}_k(x)$ by the minimization

$$\tilde{\mu}_k(x) = \arg \min_{u \in U(x)} \tilde{Q}_k(x,u).$$

Unfortunately, this method suffers from the excessive computational overhead of the Monte Carlo simulation. We are thus motivated to consider approximations that involve reduced overhead, and yet capture the essense of the basic rollout idea. We describe next an approximation approach of this type, and in the following section, we discuss its application to stochastic scheduling problems.

Approximation Using Scenaria

Let us suppose that we approximate the cost-to-go of the base policy π using *certainty equivalence*. In particular, given a state x_k at time k, we fix the remaining disturbances at some nominal values $\overline{w}_k, \overline{w}_{k+1}, \ldots, \overline{w}_{T-1}$, and we generate the associated state and control trajectory of the system using the base policy π starting from x_k and time k. The corresponding cost is denoted by $\tilde{J}_k(x_k)$, and is used as an approximation to the true cost $J_k(x_k)$. The approximate rollout control based on π is given by

$$\tilde{\mu}_k(x) = \arg \min_{u \in U(x)} E\big\{g(x_k,u,w) + \tilde{J}_{k+1}\big(f(x_k,u,w)\big)\big\}.$$

We thus need to run π from all possible next states $f(x_k,u,w)$ and evaluate the corresponding approximate cost-to-go $\tilde{J}_{k+1}\big(f(x_k,u,w)\big)$ using a single state-control trajectory calculation based on the nominal values of the uncertainty. The nominal disturbance sequence $\{\overline{w}_k, \overline{w}_{k+1}, \ldots, \overline{w}_{T-1}\}$ may be state-dependent, and in a practical setting, its choice is intended to capture "interesting and representative" aspects of the problem's uncertainty. This is hard to characterize precisely in general, but it may be meaningful in specific contexts.

The certainty equivalent approximation involves a single nominal trajectory of the remaining uncertainty. To strengthen this approach, it is natural to consider multiple trajectories of the uncertainty, called *scenaria*, and to construct an approximation to the relevant Q-factors that involves, for every one of the scenaria, the cost of the base policy π. Mathematically, we assume that we have a method, which at each state x_k, generates M uncertainty sequences

$$w^m(x_k) = (w_k^m, w_{k+1}^m, \ldots, w_{T-1}^m), \qquad m = 1, \ldots, M.$$

The sequences $w^m(x_k)$ are the scenaria at state x_k. The cost $J_k(x_k)$ of the base policy is approximated by

$$\tilde{J}_k(x_k, r) = r_0 + \sum_{m=1}^{M} r_m C_m(x_k), \tag{4.6}$$

where $r = (r_0, r_1, \ldots, r_M)$ is a vector of parameters to be determined, and $C_m(x_k)$ is the cost corresponding to an occurence of the scenario $w^m(x_k)$, when starting at state x_k and using the base policy. We may interpret the parameter r_m as an "aggregate weight" that encodes the aggregate effect on the cost-to-go function of uncertainty sequences that are similar to the scenario $w^m(x_k)$. We will assume for simplicity that r does not depend on the time index k or the state x_k. However, there are interesting possibilities for allowing a dependence of r on k and/or x_k, with straightforward changes in the following methodology. Note that, if $r_0 = 0$, the approximation (4.6) may be also be viewed as *limited simulation approach*, based on just the M scenaria $w^m(x_k)$, and using the weights r_m as "aggregate probabilities."

Given the parameter vector r, and the corresponding approximation $\tilde{J}_k(x_k, r)$ to the cost of the base policy, as defined above, a corresponding approximate rollout policy is determined by

$$\tilde{\mu}_k(x) = \arg \min_{u \in U(x)} \tilde{Q}_k(x, u, r), \tag{4.7}$$

where

$$\tilde{Q}_k(x, u, r) = E\{g(x, u, w) + \tilde{J}_{k+1}\ f(x, u, w), r)\} \tag{4.8}$$

is the approximate Q-factor. We envision here that the parameter r will be determined by an off-line "training "process and it will then be used for calculating on-line the approximate rollout policy as above.

One may use standard methods of NDP to train the parameter vector r. In particular, we may view the approximating function $\tilde{J}_k(x_k, r)$ of Eq. (4.6) as a linear feature-based architecture where the scenaria costs $C_m(x_k)$ are the features at state x_k. One possibility is to use a straightforward least squares fit of $\tilde{J}_k(x_k, r)$ to random sample values of the cost-to-go $J_k(x_k)$. These sample values may be obtained by Monte-Carlo simulation, starting from a representative subset of states. Another possibility is to use Sutton's TD(λ). We refer to the books by Bertsekas and Tsitsiklis [BeT96] and Barto and Sutton [BaS98], and the survey by Barto et. al. [BBS95] for extensive accounts of training methods and relating techniques.

We finally mention a variation of the scenario-based approximation method, whereby *partial* scenaria are used. In particular, only a portion of the future uncertain quantities are fixed at nominal scenario values, while the remaining uncertain quantities are explicitly viewed as random. The cost of scenario m at state x_k is now a random variable, and the quantity $C_m(x_k)$ used in Eq. (4.6) should be the *expected* cost of this random variable. This variation is appropriate and

makes practical sense as long as the computation of the corresponding expected scenaria costs $C_m(x_k)$ is convenient.

5. ROLLOUT ALGORITHMS FOR STOCHASTIC QUIZ PROBLEMS

We now apply the rollout approach based on certainty equivalence and scenaria to variants of the quiz problem where there is no optimal policy that is open-loop, such as the situations (e)-(i) given in Section 1. The state after questions $i_1, \ldots, i_k$ have been successfully answered, is the current partial schedule $(i_1, \ldots, i_k)$, and possibly the list of surviving quiz takers [in the case where there are multiple quiz takers, as in variant (g) of Section 1]. A (partial) scenario at this state corresponds to a (deterministic) sequence of realizations of some of the future random quantities, such as:

(1) The list of turns that will be missed in answer attempts from time k onward; this is for the case of variant (e) in Section 1.

(2) The list of new questions that will appear and old questions that will disappear from time k onward; this is for the case of variant (f) in Section 1.

(3) The specific future times at which the surviving quiz takers will drop out of the quiz; this is for the case of variant (g) in Section 1.

Given any scenario of this type at a given state, and a base heuristic such as an index or a greedy policy, the corresponding value of the heuristic [cf. the cost $C_m(x_k)$ in Eq. (4.6)] can be easily calculated. The approximate value of the heuristic at the given state can be computed by weighing the values of all the scenaria using a weight vector r, as in Eq. (4.6). In the case of a single scenario, a form of certainty equivalence is used, whereby the value of the scenario at a given state is used as the (approximate) value of the heuristic starting from that state. In the next section we present computational results for the case of a problem, which is identical to the one tested in Section 3, but a turn may be missed with a certain probability.

6. COMPUTATIONAL EXPERIMENTS WITH STOCHASTIC QUIZ PROBLEMS

6. Computational Experiments with Stochastic Quiz Problems

The class of quiz problems which we used in our computational experiments are similar to the problems used in Section 3, with the additional feature that an attempt to answer a question can be blocked with a prespecified probability, corresponding to the case of variant (e) in Section 1. The problems involve 20 questions and 20 time periods, where each question has a prescribed set of times where it can be attempted. The result of a blocking event is a loss of opportunity to answer any question at that stage. Unanswered questions can be attempted in future stages, until a wrong answer is obtained.

In order to evaluate the performance of the base policy for rollout algorithms, we use a *single partial scenario* version of the approach described in the preceding section. Assume that the blocking probability is denoted by P_b. For an M-stage problem, at any stage k, we compute an "equivalent" scenario duration T_e as the smallest integer greater than or equal to the expected number of remaining stages where there will be no blocking. The number of remaining stages is $M \quad k$, and the probability of no blocking in each one of them is $1 \quad P_b$, so we have

$$T_e = \lceil (1 \quad P_b) * (M \quad k) \rceil$$

At a given state and stage k, the expected reward of a base heuristic for the stochastic quiz problem is approximated, using Eq. (2.1), as the expected reward obtained using the heuristic in a *deterministic* quiz problem starting with the given state, with remaining duration T_e (rather than $M \quad k$).

As in Section 4, we used seven algorithms in our experiments:

(1) The optimal stochastic dynamic programming algorithm.

(2) The greedy heuristic, where questions are ranked in decreasing $p_i v_i$, and, for each stage k, the feasible unanswered question with the highest ranking is selected.

(3) The index heuristic, where questions are ranked by decreasing $p_i v_i/(1 \quad p_i v_i)$, and for each stage k, the feasible unanswered question with the highest ranking is selected.

(4) The one-step rollout policy based on the greedy heuristic and certainty equivalence policy evaluation, where, at each stage k, for every feasible unanswered question i_k and prior sequence $i_1, \ldots, i_{k \quad 1}$, the question is chosen according to the rollout rule (2.4). The function H uses the greedy heuristic as the base policy, and its performance is approximated by the performance of an equivalent non-blocking quiz problem as described above.

(5) The one-step rollout policy based on the index heuristic and certainty equivalence policy evaluation, where the function H in (2.4) uses the index heuristic as the base policy, and is approximated using the certainty equivalence approach described previously.

(6) The selective two-step lookahead rollout policy based on the greedy heuristic, with certainty equivalence policy evaluation corresponding to an equivalent non-blocking quiz problem with horizon described as above.

(7) The selective two-step lookahead rollout policy based on the index heuristic, with certainty equivalence policy evaluation corresponding to an equivalent non-blocking quiz problem with horizon described as above.

The problems selected for evaluation involve 20 possible questions and 20 stages, which are small enough so that exact solution using dynamic programming is possible. Associated with each question is a sequence of times, determined randomly for each experiment, when that question can be attempted. Floating point values were assigned randomly to each question from 1 to 10 in each problem instance. The probabilities of successfully answering each question were also chosen randomly, between a specified lower bound and 1.0. In order to evaluate the performance of the last six algorithms, each suboptimal algorithm was simulated 10,000 times, using independent event sequences determining which question attempts were blocked and which questions were answered correctly.

Our experiments focused on the effects of three factors on the relative performance of the different algorithms:

(a) The lower bound on the probability of successfully answering a question, which varied from 0.2 to 0.8

(b) The average percent of admissible questions at any one stage, which ranged from 10% to 50%.

(c) The probability 1 P_b that individual question attempts will not be blocked, ranging from 0.3 to 1.0.

As in Section 4, for each experimental condition, we generated 30 independent problems and solved them with each of the 7 algorithms, and evaluate the corresponding performance using 10,000 Monte Carlo runs. The average performance is reported for each condition.

The first set of experiments fixed the average percentage of admissible questions at a single stage to 10%, the probability that question attempts will not be blocked to 0.6, and varied the lower bound on the probability of successfully answering a question across four conditions: 0.2, 0.4, 0.6 and 0.8. Table 3 shows the results of our experiments. The average performance of the greedy and index heuristics in each condition are expressed in terms of the percentage of the optimal performance. The results for this experiment are very similar to the results we obtained

Minimum Probability of Success	0.2	0.4	0.6	0.8
Greedy Heuristic	54%	63%	73%	82%
Improvement by One-step Rollout	31%	26%	17%	6%
Improvement by Two-step Rollout	33%	26%	17%	6%
Index Heuristic	56%	67%	78%	84%
Improvement by One-step Rollout	30%	22%	12%	4%
Improvement by Two-step Rollout	31%	23%	12%	4%

Table 3: Performance of the different algorithms for stochastic quiz problems as the minimum probability of success of answering a question varies. The average percentage of admissible questions at a single stage and the probability that question attempts will not be blocked are fixed at 10% and 0.6, respectively. The numbers reported are percentage of the performance of the optimal dynamic programming solution achieved, averaged across 30 independent problems.

earlier for deterministic quiz problems. Without rollouts, the performance of either heuristic is poor, whereas the use of one-step rollouts can recover a significant percentage of the optimal performance. As the risk associated with answering questions decreases, the performance of the heuristics improves, and the resulting improvement offered by the use of rollouts decreases. On average, the advantage of using selective two-step rollouts is small, but this advantage can be large for selected difficult problems.

The second set of experiments fixed the lower bound on the probability of successfully answering a question to 0.2, and varied the average percent of admissible questions at any one stage across 3 levels: 10%, 30% and 50%. The results of these experiments are summarized in Table 4. As in the deterministic quiz problems, the performance of the greedy and index heuristics improves as the number of admissible questions at any one stage approaches 100%. The results also show that, even in cases where the heuristics achieve good performance, rollout strategies offer significant performance gains.

The last set of experiments fixed the lower bound on the probability of successfully answering

Problem Density	0.1	0.3	0.5
Greedy Heuristic	54%	65%	78%
Improvement by One-step Rollout	31%	23%	13%
Improvement by Two-step Rollout	33%	24%	13%
Index Heuristic	56%	74%	87%
Improvement by One-step Rollout	30%	15%	5%
Improvement by Two-step Rollout	31%	16%	5%

Table 4: Performance of the different algorithms on stochastic quiz problems as the average number of questions per period increases. The lower bound on the probability of successfully answering a question and the probability that question attempts will not be blocked are fixed at 0.2 and 0.6, respectively. The numbers reported are percentage of the performance of the optimal dynamic programming solution achieved, averaged across 30 independent problems.

a question to 0.2, focused on varying the probability 1 P_b that an attempt to answer a question at any one time is not blocked over 3 conditions: 0.3, 0.6 and 1.0. The last condition corresponds to the deterministic quiz problems of Section 3. Table 5 contains the results of these experiments. As the blocking probability increases, there is increased randomness as to whether questions may be available in the future. This increased randomness leads to improved performance of myopic strategies, as shown in Table 5. Again, the advantages of the rollout strategies are evident even in this favorable case.

The results in Tables 3, 4 and 5 provide ample evidence that rollout strategies enhance substantially the performance of heuristics for stochastic quiz problems, while maintaining polynomial solution complexity.

7. QUIZ PROBLEMS WITH GRAPH PRECEDENCE CONSTRAINTS

The previous set of experiments focused on quiz problems where questions could be attempted

Probability of Non-Blocking	0.3	0.6	1
Greedy Heuristic	73%	54%	41%
Improvement by One-step Rollout	17%	31%	34%
Improvement by Two-step Rollout	18%	33%	40%
Index Heuristic	75%	56%	43%
Improvement by One-step Rollout	16%	30%	34%
Improvement by Two-step Rollout	16%	31%	38%

Table 5: Performance of the different algorithms on stochastic quiz problems as the probability of non-blocking increases. The average percentage of admissible questions at a single stage and the lower bound on the probability of successfully answering a question are fixed 10% and 0.2, respectively. The numbers reported are percentage of the performance of the optimal dynamic programming solution achieved, averaged across 30 independent problems.

during specific time periods, with no constraints imposed on the questions which had been attempted previously. In order to study the effectiveness of rollout strategies for stochastic scheduling problems with precedence constraints, we defined a class of quiz problems where the sequence of questions to be attempted must form a connected path in a graph. In these problems, a question cannot be blocked as in the problems of Section 6, so there exists an optimal open-loop policy.

Let $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ be a directed graph where the nodes $\mathcal{N}$ represent questions in a quiz problem. Associated with each node n is a value for answering the question correctly, v_n, and a probability of correctly answering the question, p_n. Once a question has been answered correctly at node n, the value of subsequent visits to node n is reduced to zero, and there is no risk of failure on subsequent visits to node n.

The graph constrains the quiz problem as follows: a question n_1 may be attempted at stage k only if there is an arc $(n, n_1) \in \mathcal{A}$, where n is the question attempted at stage k 1. The graph-constrained quiz problem of duration N consists of finding a path $n_0, n_1, \ldots, n_N$ in the graph $\mathcal{G}$ such that n_0 is the fixed starting node, $(n_k, n_{k-1}) \in \mathcal{A}$ for all $k = 0, \ldots, N$ 1, and the path maximizes the expected value of the questions answered correctly before the first erroneous

answer.

The previous heuristic algorithms can be extended to the graph-constrained case. The greedy heuristic can be described as follows: Given that the current attempted question was n, determine the feasible questions i such that $(n, i) \in \mathcal{A}$. Select the feasible question which has the highest expected value for the next attempt $p_i v_i$. In the graph-constrained problem, it is possible that there are no feasible questions with positive value, and the path is forced to revisit a question already answered. If no feasible question has positive value, the greedy heuristic is modified to select a feasible node which has been visited the least number of times among the feasible nodes from node n. The index heuristic is defined similarly, except that the index $p_i v_i/(1 \quad p_i v_i)$ is used to rank the feasible questions.

One-step rollout policies can be based on the greedy or index heuristics, as before. Since the class of problems is similar to the deterministic quiz problems discussed earlier, it is straightforward to determine the expected value associated with a given policy. The rollout policies are based on exact evaluation of these expected values.

In the experiments below, we compare the following five algorithms:

(1) The optimal dynamic programming algorithm.

(2) The greedy policy.

(3) The index policy.

(4) The one-step rollout policy based on the greedy heuristic.

(5) The one-step rollout policy based on the index heuristic.

The first set of experiments involves problems with 16 questions and 16 stages. This problem size is small enough to permit exact solution using the dynamic programming algorithm. The questions were valued from 1 to 10, selected randomly. On average, each node was connected to 5 other nodes, corresponding to 30% density. In these experiments, the probability of successfully answering a question was randomly selected between a lower bound and 1.0, and the lower bound was varied from 0.2 to 0.8, thereby varying the average risk associated with a problem.

Table 6 summarizes the results of these experiments. The first observation is that the performance of the heuristics in graph-constrained problems is relatively superior to the performance obtained in the experiments in Section 4. This is due in part to the lack of structure concerning when questions could be attempted in the problems tested in Section 4. In contrast, the graph structure in this section provides a time-invariant set of constraints, leading to better performance. In spite of this improved performance, the results show that rollout algorithms can

improve the performance of the heuristics, to levels where the achieved performance is roughly 95% of the performance of the optimal dynamic programming algorithm, with a significant reduction in computation cost compared with the optimal algorithm.

Minimum Probability of Success	0.2	0.4	0.6	0.8
Greedy Heuristic	74%	77%	77%	84%
Improvement by One-step Rollout	20%	17%	14%	10%
Index Heuristic	84%	87%	89%	90%
Improvement by One-step Rollout	11%	9%	7%	5%

Table 6: Performance of the different algorithms on graph-constrained quiz problems as the minimum probability of success of answering a question increases. The probability of successfully answering a question was randomly selected between a lower bound and 1.0, and the lower bound was varied from 0.2 to 0.8. The numbers reported are percentage of the performance of the optimal dynamic programming solution achieved, averaged across 30 independent problems.

To illustrate the performance of rollout algorithms on larger problems, we ran experiments on graphs involving 100 questions and 100 stages. For problems of this size, exact solution via dynamic programming is computationally infeasible. The problems involved graphs with 10% density and varying risks as before. The results are summarized in Table 7. Since there is no optimal solution for reference, the results include the average improvement by the rollout strategies over the corresponding heuristics, expressed as a percentage of the performance achieved by the rollout strategies. The average improvement achieved by the rollout algorithms, as shown in Table 7, is consistent with the corresponding improvement shown in Table 6. The results indicate that rollout strategies continue to offer significant performance advantages over the corresponding heuristics. In contrast with the optimal dynamic programming algorithm, the average computation time for these problems when using rollout algorithms is a fraction of a second on a Sun HyperSparc workstation.

Minimum Probability of Success	0.2	0.4	0.6	0.8
Improvement over Greedy by One-step Rollout	28%	29%	31%	24%
Improvement over Index by One-step Rollout	13%	12%	10%	6%

Table 7: Performance improvement achieved by rollout algorithms over the corresponding heuristics on 100 question graph-constrained quiz problems as the minimum probability of success of answering a question increases. The numbers reported are percentage of the performance of the rollout algorithms, averaged across 30 independent problems.

8. CONCLUSION

In this paper, we studied stochastic scheduling problems arising from variations of a classical search problem known as a quiz problem. We grouped these variations into two classes: the deterministic quiz problems, for which optimal strategies can be expressed as deterministic sequences, and the stochastic quiz problems, for which optimal strategies are feedback functions of the problem state. For either of these classes, the computational complexity of obtaining exact optimal solutions grows exponentially with the size of the scheduling problem, limiting the applicability of exact techniques such as stochastic dynamic programming.

In this paper, we develop near-optimal solution approaches for deterministic and stochastic quiz problems that are computationally tractable based on the use of rollout algorithms. For stochastic quiz problems, we introduced a novel approach to policy evaluation, based on the use of scenaria, which resulted in polynomial complexity algorithms for obtaining near-optimal strategies. Our computational experiments show that these rollout algorithms can substantially improve the performance of index-based and greedy algorithms for both deterministic and stochastic quiz problems.

REFERENCES

[BBS95] Barto, A. G., Bradtke, S. J., and Singh, S. P., 1995. "Learning to Act Using Real-Time Dynamic Programming," Artificial Intelligence, Vol. 72, pp. 81-138.

[BTW97] Bertsekas, D. P., Tsitsiklis, J. N., and Wu, C., 1997. "Rollout Algorithms for Combinatorial Optimization," Heuristics, Vol. 3, pp. 245-262.

[BaS98] Barto, A. G., and Sutton, R., 1998. Reinforcement Learning. MIT Press. Cambridge, MA.

[BeT96] Bertsekas, D. P., and Tsitsiklis, J. N., 1996. Neuro-Dynamic Programming. Athena Scientific, Belmont, MA.

[Ros83] Ross, S. M., 1983. Introduction to Stochastic Dynamic Programming. Academic Press, N. Y.

[TeG96] Tesauro, G., and Galperin, G. R., 1996. "On-Line Policy Improvement Using Monte Carlo Search," presented at the 1996 Neural Information Processing Systems Conference, Denver, CO.

[Whi82] Whittle, P., 1982. Optimization over Time, Vol. I, Wiley, N. Y.

[Whi83] Whittle, P., 1983. Optimization over Time, Vol. II, Wiley, N. Y.

Approximate Dynamic Programming for the Solution of Multiplatform Path Planning Problems

Stephen D. Patek*, David A. Logan**, & David A. Castanon+

*Systems Engineering, U. Virginia
Charlottesville, VA. 22903-2442, USA
patek@virginia.edu

**ALPHATECH, INC.,
Burlington, MA. 01803, USA
david.logan@alphatech.com

+Electrical and Computer Eng., Boston U.
Boston, MA. 02215, USA
dac@bu.edu

ABSTRACT

We consider the problem of planning the paths of multiple vehicles in observing a battle space with the possibility of vehicle destruction. We illustrate the significant complexities that arise when stochastic effects (i.e. random vehicle destruction) are introduced into the model. Dynamic programming is the classical framework that characterizes solutions to our problem and drives the algorithmic development. While computationally expensive, the dynamic programming recursion can be employed to solve the stochastic problem directly. Similarly, dynamic programming can be used to solve various deterministic *auxiliary* problems whose solutions provide heuristic solutions to the original stochastic problem. We describe our approach and illustrate preliminary results.

1 Introduction

We consider the situation faced by a commander in the battlefield who seeks information about enemy strength at specific points on a map in order to make tactical decisions. The model for this that we develop in Section 2 takes the form of a large scale stochastic optimal control problem to which the classical framework of dynamic programming applies. Even if we remove the stochastic elements of the model, we have a very difficult optimization problem to solve. Indeed, deterministic vehicle routing problems are recognized as being among the most difficult of optimization problems to solve computationally. Stochastic effects such as random vehicle destruction severely complicate matters, and, since the methods of dynamic programming are feasible for only very small problems, we are faced with the necessity of using approximations. The approach we take in this paper is one of Neuro-Dynamic Programming [1] and, more specifically, rollout algorithms [2, 3].

In Section 2 we formally pose the stochastic multiplatform path planning problem. In Section 3, we describe the various solution methodologies that we apply in Section 4 to a specific instance of the problem. Our results indicate that, as would one expect, the qualitative nature of the optimal routing policy depends on the probabilities of vehicle destruction. In Section 5 we offer brief conclusions and provide pointers to future research.

2 Model

We now present a model that captures the inherent difficulty of multiplatform path planning with stochastic uncertainty. We suppose that there is a finite set of regions of interest, along with a finite set of geographically significant waypoints. The regions of interest and the waypoints are the positions (nodes in a graph) to which various reconnaissance platforms can be sent. We assume that the regions-of-interest and waypoints are connected by a finite number of arcs which are the relatively-safe conduits through which reconnaissance vehicles may travel, forming a graph. The commander has access to a finite number of reconnaissance vehicles which can be routed to the various nodes on the graph in order to gain as much information as possible on a finite time horizon. To keep the model simple, we assume that the reconnaissance vehicles are indistinguishable in that:

1. they respond to routing commands from the commander on a per-stage basis,
2. they are able to transmit intelligence data immediately upon encountering a region of interest,
3. they each require one unit of (discrete) time to traverse any arc on the graph,
4. they each experience independently the same risk of being shot down each time they traverse a given arc, and
5. they are all initially located at a "home base" (node of the graph) to which they must return by the end of the mission.

We assume that when a region of interest is first visited the desired information about that region is gained deterministically, regardless of which vehicle arrives. The value of the reconnaissance does not increase with the

number of vehicles that arrive, if two or more arrive simultaneously. We leave open the possibility that future revisits will also be worthwhile. The number of worthwhile visits will not be known to the commander ahead of time but will be revealed as a the mission progresses.

We now represent the model on a more formal level. Let $\mathcal{N} = \{0, 1, \ldots, N_N\}$ be the set of nodes of the graph that correspond to regions of interest and geographically significant waypoints. Let node 0 represent the "home base," from which the vehicles depart and to which they should try to return (if possible). Generically, we may treat all nodes as regions of interest, some of which might always offer zero value for visits (thus representing waypoints). For each node $i \in \mathcal{N}$, let $x_i(t) \in X_i$ represent the value-state of the node at time t (directly observed by the commander), which serves as an index determining the immediate value $v_i(x_i(t))$ of the *next* visit to node i. Because the vehicle transmits it's findings without delay, the reward is accrued immediately. We assume that each node's value-state can take on only a finite set of values, and the value-state changes stochastically according to an acyclic Markov chain every time a vehicle arrives. Let P_i be the transition probability matrix for node i's value-state transition process. Let $A(i) \subset \mathcal{N}$ be the subset of nodes j for which there is an arc from node i to j. This corresponds to the set of movement-options available to a vehicle located at node i. Let $B(i) \subset \mathcal{N}$ be the subset of nodes j for which there is an arc from j to node i. Note that we do not prohibit self-loops. That is, we may have $i \in A(i)$ (in which case it must also be true that $i \in B(i)$). Let N_A be the number of directed arcs in this graph representation of the problem. Given i and $j \in A(i)$, p_{ij} represents the probability of surviving the transition from i to j. Destruction is the only other possible outcome of the decision to move toward j. If the vehicle does not survive the transition, then it is lost at a cost of C_D. Otherwise, if a vehicle survives to the end of the mission, but only manages to make it back to node $i \in \mathcal{N}$, then a cost of $c_i \geq 0$ is incurred, where $c_0 = 0$. The commander is charged with controlling the movement of the vehicles on a time horizon of T stages and seeks to maximize the expected net value of the reconnaissance. We assume that N_V vehicles are available initially.

3 Solution Methodologies

Dynamic programming is the classical framework that characterizes solutions to the model of Section 2. One nice feature of the model is the fact that one can "build up" a solution for the case of n vehicles by first solving the (easier) problems involving $1, 2, \ldots, n-1$ vehicles, respectively. The reason for this is that the underlying controlled Markov chain is acyclic in the sense that once a vehicle is destroyed it cannot come back to life. This opens up the possibility of using *exact* solutions to endgames in a more general methodology involving approximate dynamic programming. Another interesting feature of the model is that there exist optimal policies which are "quasi-open loop" in the sense that the feedback is only important when vehicles are destroyed. Given a fixed number of remaining vehicles, the vehicles can be optimally routed by making a schedule of future transitions, and this plan would apply until the next vehicle is destroyed. Unfortunately, the formulation of such a plan will always involve accounting for the stochastic perturbations which can take place in every time period, so the inherent difficulty of computing optimal solutions remains. Still, at some point in the future, it may be possible to take advantage of this special structure in quickly computing nearly optimal solutions.

3.1 Dynamic Programming

To interpret the path planning problem as a dynamic program, we identify as the system to be controlled (1) the number and locations of all of the vehicles and (2) the value-states of all of the regions of interest. As a result, the number of possible system-states is large but finite, equal to $N_N{}^{N_V} \cdot \Pi_{i=1}^{N_N} |X_i|$, where $|X_i|$ is the number of value states for node i. Knowing the amount of time remaining in the mission and having access to the state of the system, the commander must choose from a finite number of routing commands for each of the remaining vehicles. We assume that it is feasible for the commander to consider all possible *combinations* of routing commands for the vehicles; however, in practice it is sometimes necessary to reduce the size of the control space by making decisions for the vehicles *in sequence* according to a prespecified order of vehicle importance. Once a routing command has been issued, the system transitions to a new state subject to the random perturbations that are possible (e.g. vehicle destruction and value-state transitions). The commander desires a routing policy which maximizes the expected reward over the finite time horizon of the reconnaissance mission.

In theory, a backwards dynamic programming recursion may be used to compute the optimal cost-to-go function for this problem. The recursion first computes the optimal expected cost-to-go with one stage of the mission remaining and then uses this in a recursive fashion to compute the optimal expected cost-to-go with two stages remaining. Each step of the recursion proceeds similarly:

$$J_k^*[\xi(k)] = \min_{u \in U[\xi(k)]} \mathbf{E}\{\, g[\xi(k), \xi(k+1)] + J_{k+1}^*[\xi(k+1)] \mid u \,\} \quad (1)$$

$$J_T^*[\xi(T)] = h[\xi(T)] \quad (2)$$

where

1. $\xi(k)$ is the state of the system at stage k.

2. u is a profile of routing command to the remaining vehicles at stage k.

3. $U[\xi(k)]$ is the set of combinations of routing commands available at state $\xi(k)$,

4. $g[\xi(k), \xi(k+1)]$ is the cost of vehicles destroyed in transitioning from $\xi(k)$ to $\xi(k+1)$, offset by the value of the intelligence data gained,

5. $h[\xi(T)]$ is the cost associated with the locations of all surviving vehicles at the terminal state $\xi(T)$, and

6. the conditional expectation is over all possible state transitions (i.e. to $\xi(k+1)$) from $\xi(k)$ under the control profile u.

The result of this recursion is a large lookup table which contains the optimal expected long term cost from every possible state of the system. Knowing the current state of the system $\xi(k)$ and having access to the optimal cost-to-go function allows us to pick optimal actions as the minimizers of the right hand side of Equation (1).

Optimal Policy *$[\pi^* = \{\mu_0^*, \ldots, \mu_{T-1}^*\}]$ Having access to the optimal cost-to-go functions J_k^* for $k = 0, \ldots, T-1$, an optimal action $\mu_k^*[\xi(k)]$ from state $\xi(k)$ at stage k is one that minimizes the right hand side of Bellman's equation:*

$$\mu_k^*[\xi(k)] \in \arg \min_{u \in U[\xi(k)]} \mathbf{E}\{ g[\xi(k), \xi(k+1)] + J_{k+1}^*[\xi(k+1)] \mid u \}. \quad (3)$$

(The minimum in this equation may not be unique.)

While theoretically sound, the dynamic programming recursion is too expensive computationally to be used in real-world scheduling problems with significant random uncertainty. This is due to the fact that large numbers of outcomes (transitions) are possible for each profile of routing commands and, even worse, the space of routing profiles grows combinatorially with the number of vehicles. Even a single step of the backwards recursion may be prohibitively expensive. Consequently, exact dynamic programming solutions are possible only for very small problems, as in the testbed problem of Section 4.

3.2 Heuristics via Deterministic Auxiliary Problems

Given the intractability of the path planning problem, we are obliged (in practice) to accept reasonable suboptimal routing policies for the vehicles. We begin in this section a discussion of several potentially useful heuristics for path planning. Perhaps the simplest type of heuristic to consider is one based on the solution to a deterministic auxiliary problem:

Auxiliary Problem 1 *Set the probabilities of destruction to zero and make the value-state transition dynamics deterministic, but otherwise keep the path planning problem identical to the original model. Choose a schedule for the vehicles to minimize the (deterministic) cost over the finite planning horizon.*

Notice that the resulting problem, being deterministic, is considerably easier than the original stochastic problem of Section 2. Unfortunately, the integer scheduling problem that remains is a vehicle routing problem where the vehicles are constrained to traveling along arcs of a graph (very difficult to solve computationally.) In this paper, we use dynamic programming to solve Auxiliary Problem 1, although more generally we may consider of any reasonable heuristic.

Supposing that Auxiliary Problem 1 is easy to solve, a reasonable heuristic for the original stochastic problem is to implement actions that would be optimal from the same state in the corresponding deterministic auxiliary problem.

Policy 1 *$[\pi^1 = \{\mu_0^1, \ldots, \mu_{T-1}^1\}]$ Compute the optimal policy $\bar{\pi}^* = \{\bar{\mu}_0^*, \ldots, \bar{\mu}_{T-1}^*\}$ for Auxiliary Problem 1. Set $\mu_k^1 = \bar{\mu}_k^*$ for all $k = 0, \ldots, T-1$.*

Another possibility is to think of the optimal cost-to-go functions in the dynamic programming solution to Auxiliary Problem 1 as heuristic value-function approximations for the original stochastic model.

Policy 2 *$[\pi^2 = \{\mu_0^2, \ldots, \mu_{T-1}^2\}]$ Having access to the optimal cost-to-go functions $\bar{J}_k^*$ for Auxiliary Problem 1, generated by a recursion analogous to that of Equations (1) and (2), choose each μ_k^2 such that $\mu_k^2[\xi(k)]$ is an action from state $\xi(k)$ at stage k which minimizes the right hand side of Bellman's equation, replacing the optimal cost to go function J_{k+1}^* with $\bar{J}_{k+1}^*$:*

$$\mu_k^2[\xi(k)] \in \arg \min_{u \in U[\xi(k)]} \mathbf{E}\{ g[\xi(k), \xi(k+1)] + \bar{J}_{k+1}^*[\xi(k+1)] \mid u \}. \quad (4)$$

(The minimum in this equation may not be unique.)

3.3 Rollouts Based on Exact Evaluation of Heuristic Cost

Given an arbitrary (possibly suboptimal) policy $\pi = \{\mu_0, \ldots, \mu_{T-1}\}$, it is possible (in theory) to use the dynamic programming recursion to compute the expected cost-to-go for every state of the system under that policy. By eliminating the min operation in Equation (1) and replacing $u(k)$ with $\mu_k[\xi(k)]$, we obtain the expected cost to go functions associated with π, denoted J_k^π, $k = 0, \ldots, T$. We obtain the effect of a policy iteration by "rolling it out" as follows.

Policy 3 (Exact Rollout) *$[\pi^3 = \{\mu_0^3, \ldots, \mu_{T-1}^3\}]$ Choose each μ_k^3 such that $\mu_k^3[\xi(k)]$ is an action from state $\xi(k)$ at stage k which minimizes the right hand side of Bellman's equation, replacing the optimal cost*

to go function J^{π}_{k+1} with J^{π}_{k+1}:

$$\mu^3_k[\xi(k)] \in \arg\min_{u\in\mathcal{U}[\xi(k)]} \mathbf{E}\{\, g[\xi(k),\xi(k+1)] + J^{\pi}_{k+1}[\xi(k+1)] \mid u \,\}. \quad (5)$$

(The minimum in this equation may not be unique.) This policy derives its name from the fact that, in practice, the evaluations $J^{\pi}_k[\xi(k),k]$ often are not computed precisely but are estimated through online Monte Carlo simulation of the original policy π. (Cf. [2] and [3].) Thus, the evaluations are made by "rolling-out" the die. Whenever we refer to Policy 3 in this paper, we imply that the evaluations of the base policy are exact. Of course, exact evaluations are impractical for most real-world problems, so this technique only applies for very small problems. Results of this type are presented in Section 4.

4 Experimental Results

We present in this section some computational results for the path planning problem of Section 2. We first give a precise description of the path planning scenario under consideration. We follow that with a brief summary of our experimental findings. The code that we have developed for this study implements the dynamic programming recursion for both the stochastic problem as well as Auxiliary Problem 1. It also computes (exactly) the expected cost-to-go function associated with Policy 1 and uses this to compute the (exact) cost-to-go associated with the rollout scheme of Policy 3.

4.1 An Instance of the Model

We consider an instance of the path planning model that involves two vehicles who must traverse the graph shown in Figure 1. Arcs in the graph are bidirectional, with thick lines representing risk-free transitions and thin lines representing transitions that are successful with the indicated probabilities. Nodes 5, 8, 9, 11, 12, and 13 are regions of interest, and the remaining nodes are simply geographic waypoints. Note that node 5 is the "big win" in this scenario, offering a minimum of 500 and up to 1000 points, to be determined immediately after the first visit. We impose a relatively short time horizon, between 8 and 20 stages, with a 200 point penalty for each vehicle that fails to return home. Notice that the circuits $\{0,1,2,3,4,5,4,3,2,1,0\}$ and $\{0,6,7,10,11,12,13,10,7,6,0\}$ each involve 10 transitions. Thus, it is not clear a priori whether

1. to send the vehicles on separate tracks (a naive attempt to "mop up" most of the value),
2. to send vehicles along redundant tracks (to maximize the probability of making it to the big win), or
3. to send vehicles out with some other routing philosophy in mind.

Notice that we allow self-loops only at nodes 0 and 5. In our experimental results we adjust the values of the parameters (p_1, p_2, p_3, p, and T) to see the qualitative nature of the various solutions. In defining Auxiliary Problem 1 for this scenario, we set $p_1 = p_2 = p_3 = 0$ and $p = 1$.

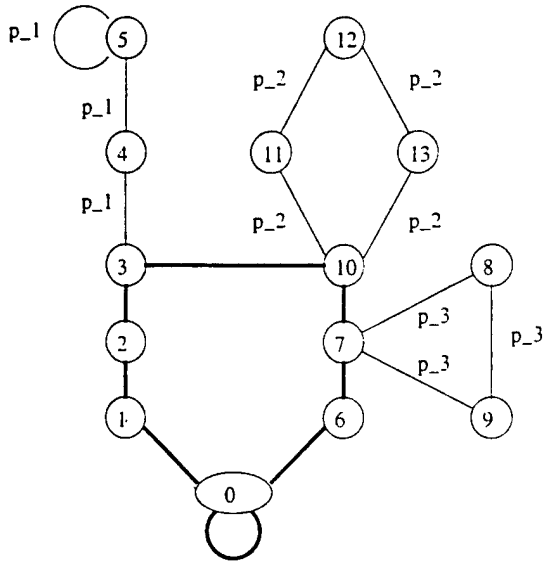

Figure 1: A scenario for the path planning problem. Note that all arcs are bidirectional. The thick arcs are risk-free, whereas the narrow arcs are traversed safely with the indicated probabilities ($p_1 < p_2 < p_3$). First visits to nodes 8 and 9 are worth 50 points; subsequent visits are worth zero. First visits to nodes 11 and 13 are worth 75 points; subsequent visits are worth zero. The first visit to node 12 is worth 150 points, and subsequent visits are worth zero. The first visit to node 5 is worth 500 points. The second visit to node 5 is worth 500 points with probability p or 0 points with probability $1-p$, and all subsequent visits are worth 0 points. Vehicles that are shot down or don't return home cost 200 points each.

4.2 Results

Table 1 illustrates the relative performance of policies π^*, π^1, π^2, and π^3 for a number of different settings of the parameters p, p_1, p_2, p_3, and T. We consider values of T equal to 8, 12, 15, and 20, and for each value of T we consider

1. a "risky" case with $p_1 = .65, p_2 = .875$, and $p_3 = .975$ and
2. a relatively "risk-free" case with $p_1 = .9, p_2 = .95$, and $p_3 = .98$.

We keep the probability of a fruitful second visit to node 5 set to one half throughout (i.e. $p = .5$). The table shows the exact expected cost to go from the

	π^*	π^1	π^2	π^3
$T = 8$ "Risky"	-161.6	-148.0	-148.0	-155.7
$T = 8$ "Risk-free"	-895.7	-867.2	-624.7	-867.3
$T = 12$ "Risky"	-332.2	-284.4	-206.1	-304.1
$T = 12$ "Risk-free"	-840.6	-836.7	-824.7	-839.6
$T = 15$ "Risky"	-378.3	-301.3	-201.4	-315.3
$T = 15$ "Risk-free"	-895.7	-867.2	-624.7	-867.3
$T = 20$ "Risky"	-391.5	-301.3	-201.4	-315.3
$T = 20$ "Risk-free"	-926.3	-867.2	-624.7	-867.3

Table 1: Table of expected costs-to-go for various time horizons T, risk levels, and policies.

initial state (all vehicles at home at time zero with all regions unvisited). We notice the following trends.

1. The naive policy π^1, which always implements an optimal action in the Auxiliary Problem 1, often does significantly worse than the optimal policy, especially in the risky cases.
2. The value-based policy π^2, which uses the optimal cost of Auxiliary Problem 1 as an approximation of the optimal cost-to-go function always does worse than π^1, often significantly worse. Perhaps the reason for this is that the approximation is state-insensitive. That is, if there is enough time left in the deterministic approximation, all of the value to be gained in the graph can be picked up with certainty.
3. The (exact) rollout policy π^3 recovers some (but not all of the) expected value achieved by π^*. The relative amount of recovery seems to diminish as the time horizon T increases.

In Figure 2 we show the nominal optimal and π^1 vehicle trajectories for the "risky", $T = 12$ case. (A nominal trajectory is the sequence of pairs of nodes visited by the vehicles given that neither of the vehicles are shot down. Our view is that these ideal trajectories indicate the *character* of the various policies under consideration.) We point out that the ideal trajectory of π^1 can be characterized by the "divide and conquer" philosophy, whereas π^* involves some redundancy (with vehicles following each other to ensure that at least some intelligence data is obtained). This philosophical dichotomy is even more pronounced in the other "risky" cases.

Generally speaking, there is insight to be gained in studying the ideal trajectories for each of the policies.

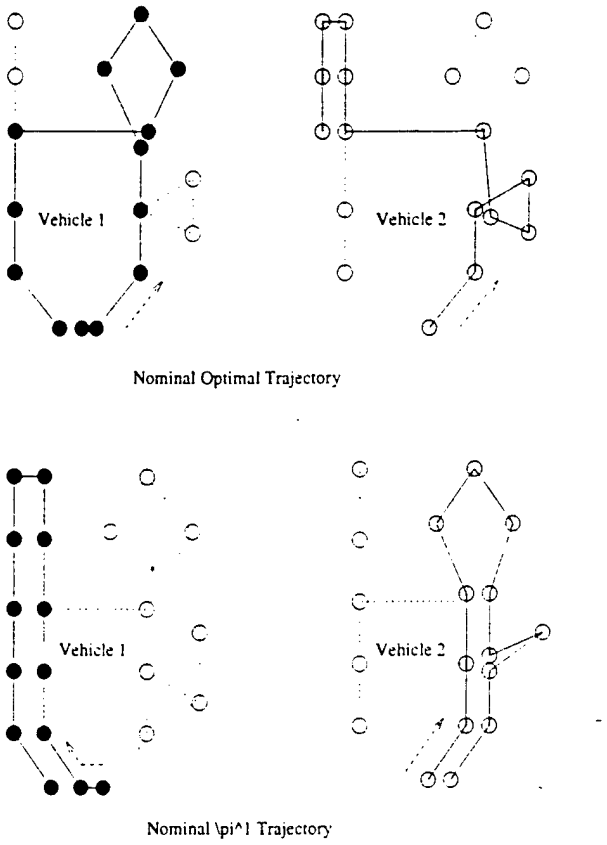

Figure 2: Nominal trajectories for π^* and π^1 for the "risky" case with $T = 12$ (Case 3).

For example, although it isn't apparent in Table 1, the total cost of the ideal trajectory of π^* for Case 1 is -900, whereas the corresponding total cost for π^1 is -950 (apparently better than optimal). The reason for this discrepancy, of course, is that the π^1 policy fails to account for the risk of being shot down along some of the arcs in its nominal trajectory.

5 Discussion and Conclusions

Unfortunately, even though the Auxiliary Problem 1 is less involved than the original stochastic problem, the computational requirements of brute-force dynamic programming for its solution are roughly equivalent. For truly large scale problems it is essential to define alternative auxiliary problems (and corresponding heuristic policies) that admit efficient computational solutions.

5.1 A Revised Model

By eliminating some of the generality in the model of Section 2, one can develop alternative auxiliary problems that are essentially large-scale network-flow problems for which rapid computational solutions are possible. For example, if value accrues deterministically

at most once for each node, then each node has at most two value-states, resulting in finite state space with $N_V{}^{N_V} \cdot 2^{\bar{N}_N}$ elements, where $\bar{N}_N$ is the number of nodes offering reconnaissance value. Consider now a network-flow model of the scheduling problem whose optimal value approximates the expected cost-to-go for the original stochastic problem for any given state at any given stage. Such a model is possible

1. by splitting each physical node into two distinct nodes (one which offers value but whose input arcs are capacity constrained and the other which offers no value but can be revisited often)

2. by making as many copies of the new graph as there are time periods remaining in the problem, so that each arc of the resulting network corresponds to either value-flow or valueless-flow for a particular vehicle *at a particular time*, and

3. by "siphoning-off" flow proportional to the probability of vehicle destruction.

Since efficient LP solvers are available for integer-relaxations of such problems, there is the hope that fast heuristic policies can be derived based on such approximations, analogous to Policy 2. (Such policies may then serve as base-policies in a rollout scheme, analogous to Policy 3.)

In implementing the scheme of the preceding paragraph, we get a new auxiliary problem with $2N_V N_N N_A T$ variables and $N_V T(N_A + N_N) + N_N$ constraints. Because the resulting model has extra side constraints (coupling the flows of all of the remaining vehicles) and gains (that account for the probability of vehicle destruction), we are compelled to use a generic LP solver, such as CPLEX. One nice feature of CPLEX is that it allows the user to input an initial basis for the primal simplex algorithm it implements. Since we will be solving many slightly different problems in the context of rollouts, we will be able to use the optimal solution of one problem as the initial basis for the next problem. (These "hotstarts" have the potential to really speed up our heuristics.)

Numerical experimentation with rollout policies based on this type of auxiliary problem are underway, and we offer only very preliminary insights. First, because our network flow model has gains and involves multicommodity side constraints, we generally get fractional solutions to the auxiliary problem. Moreover, the fractional solutions often correspond to vehicles visiting nodes for value more than once, making the approximations very optimistic (with an adverse affect on the quality of the corresponding rollout policies). It turns out to be very difficult to express side constraints that completely capture the restriction that only one vehicle can pick up value at a region of interest.

5.2 Conclusions

The multiplatform path planning problem of Section 2 is clearly very difficult to solve, despite its special structure. This is due to the fact that the cardinality of both the state and control spaces grow very rapidly with the numbers vehicles and regions of interest. In the experimental results of Section 4, we saw that the introduction of random vehicle destruction has a big effect on the qualitative nature of optimal routing solutions. Unfortunately, we were able to obtain these results only for a very small instance of the path planning problem. Our attempts to implement a reasonable and fast heuristic that will scale well with the size of the problem are currently best characterized as "work in progress." There are definitely some interesting possibilities.

Acknowledgments

The authors would like to thank D. P. Bertsekas for many helpful insights in performing this work.

REFERENCES

[1] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*, Athena Scientific, Belmont, MA, 1996.

[2] D. P. Bertsekas, J. N. Tsitsiklis, and C. Wu, "Rollout Algorithms for Combinatorial Optimization," J. of Heuristics, Vol. 3, 1997, pp. 245-262.

[3] D. P. Bertsekas and D. A. Castanon, "Rollout Algorithms for Stochastic Scheduling Problems," Report LIDS-P-2413, Lab. for Info. and Decision Systems, M.I.T., May 1998. To appear in J. of Heuristics.

Adaptive Multi-platform Scheduling in a Risky Environment*

Dimitri P. Bertsekas
Dept. of Electrical Engineering and Computer Science, M.I.T., Cambridge, MA 02139

Michael L. Curry
ALPHATECH, Inc. 50 Mall Road Burlington, MA 01803

David A. Castañon
Dept. of Electrical and Computer Engineering,Boston University, Boston, MA 02215

David Logan
ALPHATECH, Inc. 50 Mall Road Burlington, MA 01803

Abstract

In this paper, we investigate the use of rollout algorithms for adaptive multi-platform scheduling in a risky environment. The underlying decision problem is motivated by several Air Force applications: data collection, sensor management, and air operations planning. These problems may be solved optimally with stochastic dynamic programming (SDP), but have overwhelming computational requirements. Rollout algorithms reduce computational requirements by using on-line learning and simulation to approximate SDP with a base heuristic. While they do not aspire to optimal performance, rollout algorithms typically result in a consistent and substantial improvement over the underlying heuristics. A multi-platform planning and scheduling problem is used to demonstrate rollout performance.

1 Introduction

The planning and execution of multiple missions in the presence of risk is a problem which arises in many important military contexts. In data collection applications, multiple UAV platforms may be tasked to interrogate different areas, with the risk of platform destruction as each platform pursues its collection mission. In attack air operations, multiple platforms follow risky trajectories to attack enemy targets. For both applications, sensors and communication equipment can provide up-to-date information concerning individual mission and platform status, and thus provide notification of platform losses. This creates opportunities for retasking surviving platforms in order to best achieve mission objectives.

In mathematical terms, the above class of problems can be viewed as a sequential decision problem, where each decision is based on the observation of certain discrete events. These decisions affect the evolution of a system state (mission), which is also influenced by random discrete events (e.g. platform destruction). The goal is to select the current decisions as a function of the current system state, in a manner that optimizes mission performance.

The above class of problems can be formulated as Markov decision problems [3],[5]. The principal approach for solving such problems is dynamic programming (DP), which selects feedback rules to determine optimal controls for each possible state. These optimal controls are determined by evaluating at each stage the immediate expected cost of the current decision, plus the future optimal cost-to-go over future decisions. However, it is well known that computation of the optimal cost-to-go for each future state is computationally intractable for all but the simplest of problems, making direct application of DP an impossible task for multi-platform control.

In recent years, there has been a great deal of research on approximate DP methods based on computing suitable approximations to the optimal cost-to-go. These methods are collectively known as neuro-dynamic programming (NDP) [1]. In NDP, the optimal cost-to-go is approximated by a parametric function; critical issues for NDP include the selection of the parametric class of approximating functions, and selection of the approximating parameters.

In this paper, we apply a particular class of NDP algorithms, known as rollout algorithms [2], to risky multi-platform planning and scheduling problems. Rollout algorithms are a form of NDP which exploits

* This work was supported by the Air Force Office of Scientific Research under contract #F49620-98-C-0023.

knowledge of suboptimal heuristic decision rules to obtain approximations to the optimal cost-to-go for use in NDP. We develop different rollout algorithms for risky multi-platform scheduling, and illustrate the relative performance of the rollout algorithms and the original suboptimal decision rules in the context of a specific example. The results illustrate that significant performance improvements can be obtained using rollout algorithms, with a modest increase in computation complexity.

2 Illustrative Overview

To illustrate the types of problems of interest and results developed in this paper, consider the data collection problem illustrated in Figure 1. There are several data collection assets, which may travel to examine targets. There is a value associated with collecting the information on each target. Platforms also run the risk of destruction while performing collection on a asset, due to the presence of local defenses.

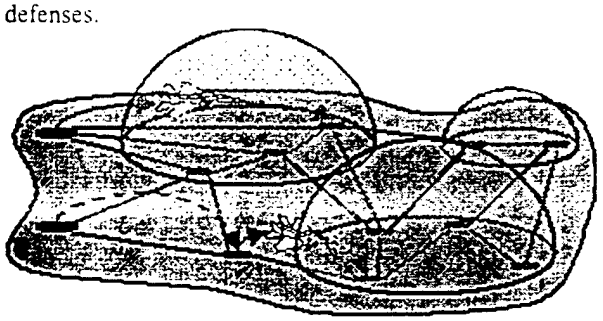

Figure 1 Illustration of Data Collection Problem

Ideally, each data collection asset will be provided a schedule of targets for information collection, which is coordinated among assets to ensure maximal value collected. However, due to the risk inherent in the collection process, platforms can be destroyed, and thus the original schedules should be adapted whenever a destruction event occurs in order to recover the most collection value. If these abrupt events are not anticipated in the original schedules, the possible modifications to the schedules may be so constrained that highly sub-optimal performance results.

The basic theory of dynamic programming provides a framework for developing schedules which anticipate the future occurrence of contingencies such as platform destruction, and hedge the selected schedules in anticipation of needed retasking. Thus, the resulting schedules can be adapted to contingencies with minimal performance degradation, resulting in robust, stable control.

The computational requirements of DP depend on the number of future states required to describe the system. To illustrate the number of states required, assume that there are N targets, M collection assets, and that we simplify physical position descriptions to describe only the N positions of the targets. Then, the number of possible combinations of positions is M^N, and the number of possible uncollected target sets at a given time is 2^N, resulting in numbers of states $(2M)^N$. For modest numbers of assets and targets, the number of states far exceeds our capability for computing and/or storing the resulting optimal decision rules.

Using NDP principles such as rollout strategies greatly reduces the resulting computational complexity. DP considers all of the possible states and computes a tentative decision for each possible state, whereas NDP only computes decisions for states that actually occur in the scenario. Thus, the number of states considered by NDP considered is much smaller, but can only be determined in real-time. In the rollout methodology, once the scenario reaches a given state where a contingency has been observed, new plan options are evaluated in real-time to select the future actions. The result is a practical algorithm for feedback control in complex multi-platform planning and scheduling applications. The fundamental questions about this approach are how good is the performance achieved, and how much real time computation is required. These questions are explored in greater detail in the subsequent sections.

3 Rollout Algorithms

Consider a discrete-time version of a dynamic decision problem,

$$x_{k+1} = f(x_k, u_k, \omega_k)$$

where x_k is the state, u_k is the control to be selected from a finite set $U(x_k)$, and ω_k is a random disturbance. Denote the single-stage cost of control u from state x and disturbance ω by $g(x,u,\omega)$.

A control policy $\pi = \{\mu_0, \mu_1, \cdots\}$ maps, for each stage k, a state x to a control value $\mu_k(x) \in U(x)$. In the N-stage horizon problems considered herein, k takes values $0,1,\cdots,N-1$, and there is also terminal cost $G(x_N)$ that depends on the terminal state x_N. The cost-to-go of policy π starting from a state x_k at time k can be computed using the following DP recursion

$$J_k^\pi(x) = E\{g(x,\mu_k(x),\omega) + J_{k+1}^\pi(f(x,\mu_k(x),\omega))\} \qquad (1)$$

for all k and with the initial condition

$$J^{\pi}{}_{N}(x) = G(x)$$

The rollout policy based on π is denoted by $\overline{\pi} = \{\overline{\mu}_0, \overline{\mu}_1, \cdots\}$, and is defined by the operation

$$\overline{\mu}_k(x) = \arg\min_{u \in U(x)} E\{g(x,u,\omega) + J^{\pi}_{k+1}(f(x,u,\omega))\} \qquad (2)$$

for all x and k. Thus the rollout policy selects decisions by balancing the current cost with future costs-to-go, where the optimal costs-to-go are approximated by the performance of the base policy π.

A straightforward approach for computing the rollout control at a given state x and time k is to use Monte Carlo simulations of the base policy. To implement this approach, we consider all possible controls $u \in U(x)$ and generate a "large" number of simulation trajectories of the system starting from x, using u as the first control, and using the policy π thereafter. Thus the simulated trajectory has the form

$$x_{i+1} = f(x_i, \mu_i(x_i), \omega_i) \quad i = k+1, \cdots N-1$$

where the first generated state is

$$x_{k+1} = f(x, u, \omega_k)$$

The costs corresponding to these trajectories are averaged to obtain the Q-factor

$$Q(x,u) = E\{g(x,u,\omega) + J^{\pi}{}_{k+1}(f(x,u,\omega))\}$$

In reality, only an approximation $\tilde{Q}(x,u)$ is obtained because of the associated simulation error. The approximation becomes increasingly accurate as the number of simulation trajectories increases. Once the approximate Q-factor $\tilde{Q}(x,u)$ corresponding to each control $u \in U(x)$ is computed, we obtain the approximate rollout control $\tilde{\mu}_k(x)$ by the minimization

$$\tilde{\mu}_k(x) = \arg\min_{u \in U(x)} \tilde{Q}_k(x,u)$$

4 Example: Data Collection Problem

The graph in Figure 2 corresponds to an example data collection problem. Each node represents a geographical area of interest with a one-time value (i.e., data may only be collected once from each location). The arcs represent connectivity among the geographical regions and may be successfully traversed with a known probability. Platforms traverse the graph and collect data (value) at each node, or else they are destroyed while traversing specific arcs. If a platform is destroyed on an arc, the value of the destination node is not collected, which can result in retasking other platforms.

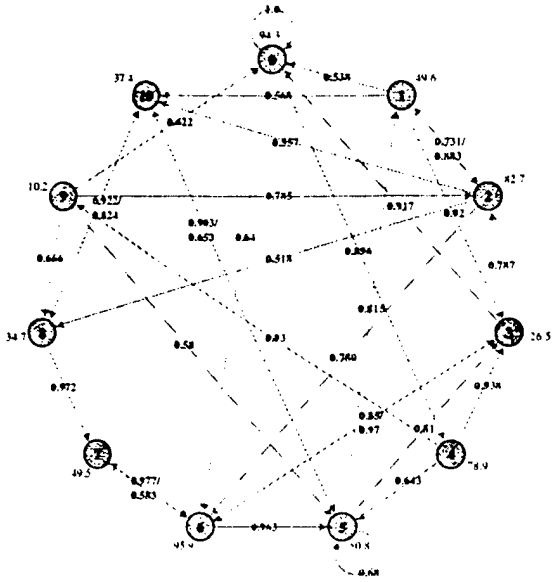

Figure 2 Graph Representation of the Data Collection Problem

The objective is to control the platforms in order to maximize the expected total value collected after N stages ($N = 10$ will be used). Each platform begins at a base node (in this case, node 0 for all platforms) and may traverse one arc during each stage. If a platform does not return to its base node within N stages, there is a penalty associated corresponding to platform loss.

4.1 The Base Policy: Greedy

As a base policy for rollout, we use the greedy policy $\pi = \{\mu_0, \mu_1, \cdots\}$, which is defined by the operation

$$\mu_k(x) = \arg\max_{u \in U(x)} E\{g(x,u,\omega)\}$$

for all x and k. The control u is a vector of locations corresponding to the next destination of each platform. Similarly, each element of $\mu_k(x) = [\mu^0_k(x), \mu^1_k(x), \cdots]$ corresponds to a specific platform.

To reduce the computational overhead, we consider the platforms sequentially. The control for the first platform, $\mu^0_k(x)$, is selected independent of the other platforms' controls as:

$$\mu^0_k(x) = \max_{u \in U^0(x)} E\{g(x,u,\omega)\}$$

where $U^0(x)$ are feasible controls for platform 0. The control, $\mu^j_k(x)$, for subsequent platforms is conditioned on all the previously selected controls $\mu^0_k(x), \mu^1_k(x), \cdots, \mu^{j-1}_k(x)$ and defined by the operation

$$\mu^j_k(x) = \max_{u \in U^j(x)} E\{g(x,u,\omega) | \mu^0_k(x), \cdots, \mu^{j-1}_k(x)\}$$

This allows the greedy policy to anticipate the arrival of platforms at specific nodes based on previously selected controls.

The greedy policy also forces platforms to return within N stages by constraining the set $U_k(x)$ of feasible controls to those for which a return within N stages is possible

The performance of the greedy policy corresponds to the cost-to-go from the initial state x_0.

$$J^{\pi}{}_0(x_0) = E\left\{G(x_N) + \sum_{i=0}^{N-1} g(x_i, \mu_i(x_i), \omega_i)\right\}$$

where the expectation is taken over simulation trajectories of the form

$$x_{i+1} = f(x_i, \mu_i(x_i), \omega_i) \quad i = 0, \cdots, N-1$$

The performance of the greedy policy provides a baseline for evaluating the rollout policy.

4.2 Rollout Algorithm

The rollout policy is computed using the greedy policy as its base policy, as indicated in equations (1-2). The performance of the rollout policy is evaluated in a manner similar to the greedy policy, by using the cost-to-go from the initial state x_0.

$$J^{\bar{\pi}}{}_0(x_0) = E\left\{G(x_N) + \sum_{i=0}^{N-1} g(x_i, \bar{\mu}_i(x_i), \omega_i)\right\}$$

with the simulation trajectories

$$x_{i+1} = f(x_i, \bar{\mu}_i(x_i), \omega_i) \quad i = 0, \cdots N-1$$

To reduce the relative variance of performance values, we use the same simulation trajectories in the evaluations of all policies.

One drawback of this approach is that many on-line Monte Carlo simulations may be required to compute the rollout decision at a state. As an alternative, we can use approximations trained with off-line simulations, as discussed in the next subsection.

4.3 Rollouts and Neural Approximations

To reduce the on-line computational overhead of the rollout policies, we propose to train off-line a parametric approximation of the greedy policy performance based on features which characterize the current state. In particular, the features that we use correspond to the values achieved by the greedy policy under a small number of certainty-equivalence scenarios, which capture the graphical dependence of the scheduling problem. This approach was initially proposed in [2].

To compute a feature at a given state x_k at time k, we fix the remaining disturbances at some nominal values $\bar{\omega}_k, \bar{\omega}_{k+1}, \cdots, \bar{\omega}_{N-1}$, and generate a state and control trajectory of the system using the base policy π starting from x_k and time k. The corresponding cost is denoted by $\tilde{J}_k^{\pi}(x_k)$, and is a feature which is used to estimate the true cost $J_k^{\pi}(x_k)$. We use a small number of disturbance trajectories corresponding to different scenarios. The feature values computed for each of these scenarios are combined parametrically to approximate the cost of the base policy using the functional form:

$$\tilde{J}_k(x_k, r) = r_0 + \sum_{m=1}^{M} r_m C_m(x_k) \qquad (3)$$

where $r = (r_0, r_1, \cdots, r_M)$ is a vector of parameters to be determined, and $C_m(x_k)$ is the cost corresponding to the m^{th} scenario. The parameters r are determined by an off-line training process using simulations of the base policy. Equation (3) can then be used on-line, computing the costs $C_m(x_k)$, to evaluate the base policy cost from state x_k at time k.

5 Experimental Results

A series of experiments were performed on the example problem presented in section 4.1, evaluating the performance of the base greedy policy and different variations of rollout algorithms.

The greedy heuristic used for the baseline policy is based on an objective function with two terms, one associated with the achievable value of data collected and the other associated with the potential loss of the vehicle. These values depend on probability ratios associated with risk. The objective function for vehicle k at state x_i is given by

$$g_k(u_i, x_i, \omega) = \frac{p_{ij}}{(1-p_{ij})} n_j(x_i) - \frac{(1-p_{ij})}{p_{ij}} v_k$$

where $n_j(x)$ is the achievable value of option j given the current state, v_k is the value of vehicle k, and p_{ij} is the transition probability associated with option j (p_{ij} characterizes the disturbance ω). This objective function is a risk neutral strategy that computes the marginal difference between the largest acceptable loss and the smallest acceptable gain associated with option j.

The greedy heuristic is evaluated by determining the cost-to-go from the initial state x_0.

rollout algorithms using on-line Monte Carlo simulations. The parametric approximations suffered from two limitations: First, the training techniques often failed to identify the best weight combinations. Second, the parametric approximations were unable to generalize accurately across the broad class of states which occurred in the problem. Our experiments were limited to simple classes of parametric approximations using the concept of certainty equivalence scenarios. Exploration of alternative approximations using different features is an area for future investigations.

The main limitation of the Monte Carlo rollout algorithms is the amount of on-line computation required to evaluate the different options at each state. We are currently investigating techniques based on discrete-event systems and perturbation analysis [4] to reduce the number of simulations required to evaluate multiple alternatives.

7 References

[1] Bertsekas, D.P., Tsitsikis, J.N., *Neuro-Dynamic Programming*, Athena Scientific, 1996.

[2] Bertsekas, D.P., Castañon, D.A., "Rollout Algorithms for Stochastic Scheduling Problems," *Journal of Heuristics*, V. 5, 1999.

[3] Bertsekas, D.P., *Dynamic Programming and Optimal Control*, Athena Scientific, 1995

[4] Ho, Y.C., Cassandras, C.G., Chen, C.H., Dai, L., "Ordinal Optimization and Simulation," submitted to special issue on simulation to be published by INFORMS 1999.

[5] Ross, S. M., *Introduction to Stochastic Dynamic Programming*, Academic Press, N.Y., 1983.

Dynamic Programming Methods for Adaptive Multi-platform Scheduling in a Risky Environment[1]

Dimitri P. Bertsekas
Dept. of Electrical Engineering and Computer Science, M.I.T., Cambridge, Mass., 02139

David A. Castañon
Dept. of Electrical and Computer Engineering, Boston University, Boston, Mass., 02215

Michael L. Curry, David Logan, Cynara Wu[2]
ALPHATECH, Inc., 50 Mall Road, Burlington, MA 01803

Abstract

In this paper, we investigate alternatives to simulation-based approximate dynamic programming methods for adaptive multi-platform scheduling in a risky environment. In a recent effort, we considered rollout algorithms, in which on-line simulation was found to be more reliable than off-line training. Unfortunately, a large amount of computational resources was required to run even a modest number of Monte Carlo simulations. In this paper, we consider alternatives to using simulation. The first approach consists of using limited lookahead policies, which reduce computational requirements by considering value explicitly over a limited horizon and approximating the value of the remaining stages. The second approach decomposes the problem into sub-problems corresponding to platforms. In our computational experiments, we found that many of the variations of these approaches required significantly less computation time than rollout algorithms and also obtained results that were substantially superior.

1. Introduction

The planning and execution of multiple missions in the presence of risk is a problem that arises in many important military contexts. In data collection applications, multiple UAV platforms may be tasked to interrogate different areas, with the risk of platform destruction as each platform pursues its collection mission. In attack air operations, multiple platforms follow risky trajectories to attack enemy targets. For both applications, sensors and communication equipment can provide up-to-date information concerning individual mission and platform status, and thus provide notification of platform losses. This creates opportunities for replanning, using feedback to retask surviving platforms in order to best achieve mission objectives.

In mathematical terms, the above class of problems can be formulated as Markov decision processes. At each stage of the process, decisions are made that affect the evolution of a system state, which is also influenced by random discrete events. The goal is to select the current decision as a function of the current state in order to optimize mission performance.

The principal approach for solving Markov decision problems is dynamic programming (DP). In comparing the available controls at a given state i, DP considers the current stage value, but also takes into account the desirability of the next state j. It "ranks" different states j by using, in addition to the current stage value, the optimal value (over all remaining stages) starting from j. This optimal value is denoted $J^*(j)$ and referred to as the optimal *value-to-go* of j. Unfortunately, it is well known that the computation of J^* is overwhelming for many important problems.

There has been a great deal of research on DP methods that replace the optimal value-to-go $J^*(j)$ with a suitable approximation for the purpose of comparing the available controls at each state. These methods are collectively known as neuro-dynamic programming (NDP). Previously, we applied a particular class of NDP

[1] This work was supported by the Air Force Office of Scientific Research under contract #F49620-98-C-0023.
[2] *Corresponding Author: phone (781)273-3388, fax (781)273-9345, e-mail cynara.wu@alphatech.com*

algorithms, known as rollout algorithms, to risky multi-platform planning and scheduling problems. Rollout algorithms are a form of NDP that exploit knowledge of suboptimal heuristic decision rules to obtain approximations to the optimal value-to-go. We developed several rollout algorithms for risky multi-platform scheduling, using on-line Monte Carlo simulations to evaluate the reference base heuristic policies, and found that they performed significantly better than the base policies as well as off-line training methods. However, even using a modest number of Monte Carlo simulations resulted in large computation times.

In this paper, we consider alternatives to using on-line simulations. In particular, we consider two approaches that use analytic approximations of the value function. We first consider a class of approximation techniques in which the control exercised at a state i is determined by considering the costs accumulated over several stages, and then applying an approximation to the value-to-go from the resulting states. The rollout algorithms considered in our previous effort are a special case in which a single-stage policy is employed and on-line simulation is used in combination with a base heuristic to approximate the value-to-go.

Our second approach involves exploiting the structure of the problem and decomposing the problem into sub-problems, each of which is associated with a corresponding platform. Each sub-problem is solved independently but takes into account the results of previously solved sub-problems.

The paper is organized as follows. In Section 2, we describe the data collection problem which we are addressing. In Section 3, we present the framework for limited lookahead policies. In Section 4, we describe our decomposition approach to the problem. In Section 5, we present some computational results.

2. Example Data Collection Problem

The graph in Figure 1 is an example corresponding to a data collection problem. Each node represents a geographical area of interest with a one-time value (i.e., data may only be collected once from each location). The arcs represent connectivity among the geographical regions and may be successfully traversed with a known probability. Platforms traverse the graph and collect data (value) at each node, or else they are destroyed while traversing specific arcs. If a platform is destroyed on an arc, the value of the destination node is not collected, which can result in retasking other platforms.

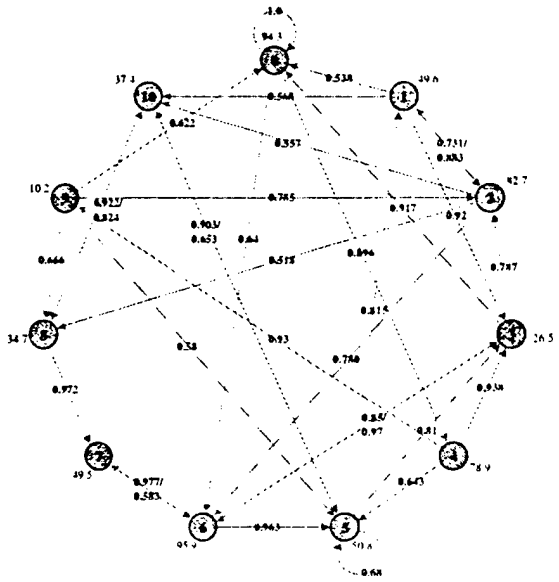

Figure 1 Graph Representation of the data collection problem.

The objective is to control the platforms in order to maximize the expected total value collected after N stages. Each platform begins at a base node (in this case, node 0 for all platforms) and may traverse one arc during each stage. There is a reward for each platform that has safely returned to its base node at the end of the Nth stage.

3. Limited Lookahead Policies

Consider a discrete-time dynamic system,

$$x_{k+1} = f_k(x_k, u_k, \omega_k),$$

where x_k is the state, u_k is the control to be selected from a finite set $U_k(x_k)$, and ω_k is a random disturbance. Denote the single-stage reward of control u from state x and disturbance ω by $g_k(x,u,\omega)$. A control policy $\pi=\{\mu_0,\mu_1,...,\mu_{N-1}\}$ maps, for each stage k, a state x_k to a control value $\mu_k(x_k)\in U_k(x_k)$. There is a terminal reward $G(x_N)$ that depends on the terminal state x_N. The value-to-go of an optimal policy $\pi^*=\{\mu_0^*,\mu_1^*,....,\mu_{N-1}^*\}$ starting from a state x_k at stage k can be computed using the following DP recursion

$$J_k^*(x_k)=\max_{u_k\in U_k(x_k)} E\{g_k(x_k,u_k,\omega_k)+J_{k+1}^*(f_k(x_k,u_k,\omega_k))\},$$

for all k and with the initial condition

$$J_N^*(x_N)=G(x_N).$$

For our problem, the state can be represented by a vector indicating for each node whether or not its value has been collected and by another vector indicating for each platform whether or not it is alive and if so, the node

at which the platform is located. The control at a particular stage provides for each platform that is alive a node that the platform is to attempt to visit during the current stage. If the platform successfully traverses the arc connecting its current node to the next node and the value of the node has not yet been collected, the current stage reward includes the value of the node. If the platform successfully reaches its base node during the last stage, there is a terminal reward associated with the platform.

Under a one-step lookahead policy, the control selected at stage k and state x_k is that which maximizes the following expression:

$$\max_{u_k \in U_k(x_k)} E\left\{g_k(x_k,u_k,\omega_k)+\tilde{J}_{k+1}(f_k(x_k,u_k,\omega_k))\right\},$$

where $\tilde{J}_{k+1}$ is some approximation of the value-to-go function J^*_{k+1}. Under a two-step lookahead policy, the control selected at stage k and state x_k is that which maximizes the above expression when $\tilde{J}_{k+1}$ is itself a one-step lookahead approximation; i.e., for all possible states $x_{k+1} = f_k(x_k,u_k,\omega_k)$, we have

$$\tilde{J}_{k+1}(x_{k+1}) = \max_{u_{k+1} \in U_{k+1}(x_{k+1})} E\left\{\begin{array}{l} g_{k+1}(x_k,u_k,\omega_k)+ \\ \tilde{J}_{k+2}(f_{k+1}(x_{k+1},u_{k+1},\omega_{k+1}))\end{array}\right\}.$$

Other multi-stage lookahead policies are similarly defined. Note that the number of lookahead stages, M, should be less than or equal to N-k-1. Essentially, the M-stage lookahead policy selects at stage k its decision by determining the optimal policy if there were only M stages remaining and the terminal cost was given by $E_{x_M}\left\{\tilde{J}_{k+M+1}(x_M)\right\}$, where x_M is the state resulting from applying the policy for the M decisions. A decision is selected, and the process is repeated at the next stage. The lookahead horizon is limited to the number of remaining stages, and so if the number of remaining stages is less than M, the M-stage lookahead policy determines the optimal strategy. A special case of such policies in which the value-to-go is approximated with zero is referred to in the literature as rolling or receding horizon procedures.

Generally, the effectiveness of limited lookahead policies depends on two factors:

1. The quality of the value-to-go approximation – performance of the policy typically improves with approximation quality.
2. The length of the lookahead horizon – performance of a policy typically improves as the horizon becomes longer (at least for small horizon lengths, e.g., 1-4).

However, as the size of the lookahead increases, the number of possible states that can be visited increases exponentially. To keep the overall computation practical, the complexity of the value-to-go approximation should be reduced for larger lookahead sizes. Balancing such tradeoffs is therefore a critical element in determining the size of the lookahead and the method for approximating the value-to-go. This paper explores several possibilities and tries to quantify the associated tradeoffs. One of the advantages of using limited lookahead policies for our particular problem is that the number of controls at a particular stage is fairly small and as a result, the computation required to explore all states that can be visited over the next M stages is manageable for small M.

3.1. Pruned Limited Lookahead Policies

Since the number of states that can be visited over M stages grows exponentially in M and also in the number of platforms, limited lookahead policies for M>1 are impractical for problems with many platforms. One approach to reducing the computation required for limited lookahead policies is to limit the number of states that can be visited. This can be accomplished by "pruning" controls that yield inferior intermediate values.

A pruned version of a limited lookahead policy depends on an integer parameter B that is typically selected through trial and error. In particular, we determine the one-step lookahead values for all controls available from our initial state. Controls that are not among those with one of the B best one-step lookahead values are pruned. We then repeat this process for each state that can be reached from a control that was not pruned and determine the one-step lookahead values for all controls available from these states. For each of these states, controls that are not among those with one of the B best one-step lookahead values are pruned. The number of times this process takes place is equal to the size of the lookahead.

Since the number of controls that are expanded from every state at every stage is limited, the computation required to find pruned policies is not exponential in the number of platforms. However, the computation is still exponential in the size of the lookahead.

4. Platform Decomposition

We now present an approach that involves exploiting the structure of our specific problem and decomposing it into a set of simpler problems. In particular, we decompose the problem into a separate sub-problem for each platform. This sub-problem consists of determining the optimal sequence of nodes, or path, to visit assuming that platform was the only one available. The optimal solution to each sub-problem can be found analytically. After a sub-problem is solved for a particular platform and before the next sub-problem is solved, the value of each

node in the associated path is updated to the value of the node multiplied by the probability that the node was not visited by the platform. This allows platforms to take into account paths assigned to previously scheduled platforms. When all of the sub-problems have been solved, a set of paths for each platform results. An outline of the platform decomposition approach is given below.

1. Assume that the platforms are ordered 1,2,...,V, and start with platform i=1.
2. Solve the single-platform problem optimally by finding a path or sequence of nodes $(n_{i1},n_{i2},...,n_{iN})$ that the platform should attempt to visit in order to maximize its expected value (which consists of collected node values plus the reward for the platform returning to the base station if n_{iN} is the base node).
3. For every node in the path obtained in (2), scale the value of the node to 1 minus the probability that the node will be visited by platform i. This allows platforms that are scheduled later to take into account the path assigned to the current platform.
4. If i is less than the number of platforms, then let i=i+1 and go to (2). Otherwise, we are done.

The single-platform problem in step 2 can be solved using dynamic programming or by exhaustively considering all possible paths with N nodes. The computation required in either case is $O(D^N)$, where N is the number of stages and D is the average degree of a node. For sparsely connected graphs, the computation required is minimal.

The set of sub-problems can be solved once for a particular ordering of platforms or multiple times for various platform orderings. We will discuss several possibilities in the next section.

The platform decomposition heuristic yields for each platform i a path $(n_{ij},n_{i(j+1)},...,n_{iN})$, where j is the stage at which the heuristic is applied. This heuristic can be applied once before the mission begins to obtain a policy in which platform i attempts to visit node n_{ij} during the jth stage if it has not yet been destroyed. The heuristic can also be applied at every stage (for platforms that are still alive) using up-to-date state information, obtaining a policy in which platform i attempts to visit node n_{ij} during the jth stage. Finally, the heuristic can also be used to compute a value-to-go approximation for limited lookahead policies.

One of the main advantages to the platform decomposition approach is that the computation required is considerably smaller than limited lookahead policies. Assuming that the number of platform orderings considered remains fixed, the computation grows linearly in the number of platforms. In addition, as will be seen below, the method obtains solutions that are very close to the optimal. Unfortunately, while limited lookahead policies generalize easily to other problems, other problems may not have structures that easily decompose into sub-problems.

5. Computational Results

We now present some computational results from applying the above approaches to the problem described in Section 2. We consider a problem with N=10 stages, and either three or four platforms. The return rewards for the platforms were set to 12.7, 17.5, 19.2, and 55.0, and the most valuable platform was not included in the three-platform problems.

5.1. Limited Lookahead Policies

A limited lookahead policy consists of two main elements: the lookahead horizon, and the approximation of the value-to-go. We vary the size of the horizon from one to three and consider a number of approximations to the value-to-go. While there is some difference in the complexity of the value-to-go approximations, each one is straightforward to compute.

In many of our approaches, the value-to-go approximation for a particular state x after the first k stages, $\tilde{J}_k(x)$, involves heuristically generating for each platform i, a path or sequence of nodes $(n_{i(k+1)},n_{i(k+2)},...,n_{iN})$ to attempt to visit during the remaining N-k stages. We denote this collection of paths $P(x,k)$. Assuming each platform attempts to visit the nodes in its path, we can determine the expected collected value resulting from visiting nodes not visited during the first k stages:

$$C[P(x,k)]=\sum_{\substack{\text{nodes } n \text{ not}\\ \text{yet visited}}}\left(1-\prod_{\text{platforms } i}(1-p_{in})\right)c_n .$$

In the above equation, c_n is the one-time value associated with node n, and p_{in} is the probability that platform i visits node n:

$$p_{in}=\begin{cases}\prod_{j=k+1}^{l-1}p(n_{ij},n_{i(j+1)}), & \text{if } n_{il}=n \text{ for some } l,\\ 0, & \text{otherwise,}\end{cases}$$

where $p(n_{ij},n_{i(j+1)})$ is the probability of successfully traversing the arc connecting nodes n_{ij} and $n_{i(j+1)}$. To understand the expression for $C[P(x,k)]$, note that the term $\prod_{\text{platforms } i}(1-p_{in})$ provides the probability that none of the platforms successfully visits node n. The term

$\left(1-\prod_{\text{platforms } i}(1-p_{in})\right)c_n$ then provides the expected collected value at node n (the probability that at least one platform successfully visits the node multiplied by the node value).

We can also determine the expected reward resulting from platforms returning to the base node:

$$R[P(x,k)]=\sum_{\text{platforms } i} q_i v_i ,$$

where

$$q_i=\begin{cases}\prod_{j=k+1}^{N-1} p(n_{ij},n_{i(j+1)}), & \text{if } n_N \text{ is the base node,}\\ 0, & \text{otherwise,}\end{cases}$$

is the probability that platform i returns to the base node and v_i is the platform return reward.

The approximations to the value-to-go that we consider are given below. As can be seen in the descriptions, many of the approximations involve a combination of the expected collected node value, $C[P(x,k)]$, and the expected platform return reward, $R[P(x,k)]$, assuming each platform attempts to visit the nodes in the paths specified in $P(x,k)$.

1. The first approach approximates the value-to-go with zero:

   $$\tilde{J}_k(x)=0.$$

2. The second approach approximates the value-to-go with the sum of the expected collected node value and the expected platform return reward collected over a set of greedy paths:

   $$\tilde{J}_k(x)=C[P_g(x,k)]+R[P_g(x,k)].$$

   The nodes along the greedy path for platform i, $(n_{i(k+1)},\ldots,n_{iN})$, are determined as follows:

   $$n_{i(j+1)}=\arg\max_{n\in\eta(n_{ij})}\{p(n_{ij},n)\cdot c_n\},$$

   where $\eta(n_{ij})$ is the set of nodes that can be reached from node n_{ij}, and n_{ik} is the node at which platform i is located after k stages.

3. The third approach approximates the value-to-go with the expected platform return reward collected over the set of "safest" paths:

   $$\tilde{J}_k(x)=R[P_s(x,k)].$$

   The safest path is that which yields the highest probability of a platform returning successfully to its base node. These paths can be computed apriori using dynamic programming. (Essentially, the computation is equivalent to solving a set of shortest path problems.)

4. The fourth approach approximates the value-to-go with the sum of the expected collected node value and the expected platform return reward collected over the set of safest paths:

   $$\tilde{J}_k(x)=C[P_s(x,k)]+R[P_s(x,k)].$$

5. The fifth approach approximates the value-to-go with the sum of the expected collected node value and the expected platform return reward collected over the set of "most valuable" paths:

   $$\tilde{J}_k(x)=C[P_m(x,k)]+R[P_m(x,k)].$$

   The most valuable path is that which yields the highest expected total value that could be attained by a single vehicle during the remaining stages assuming none of the values at any of the nodes have yet been collected. These paths can also be computed apriori using dynamic programming.

6. The sixth approach combines (4) and (5). The value-to-go is approximated with the maximum of the values determined by those approaches.

Table 1 provides the expected optimal values for the problem illustrated in Figure 1 for a three-platform problem and a four-platform problem. We have computed these values using dynamic programming, and the computation required for the four-platform problem was approximately one week on a Sun Ultra 60 workstation. Table 1 also provides the results of applying a greedy algorithm, in which each platform selects as its next node that which maximizes its expected collected value for that stage, to one thousand sample trajectories. The performance achieved in our earlier efforts of applying rollout strategies using 20 or more Monte Carlo simulations ranged on average from 600 to 610 for the four-platform problem.

Table 1 The expected optimal values and the results of applying the greedy algorithm for the three and four platform problems.

# Platforms	Expected Optimal	Greedy
Three	574.5	475.72
Four	641.0	533.89

Tables 2 and 3 provide the values averaged over one thousand sample trajectories by applying the limited lookahead polices for lookahead sizes of one to three, using the six value-to-go approximations described above. The particular approximation approach used is given in the leftmost column. As can be seen, while the 2-stage policies generally provided results that improved significantly upon those of the 1-stage policies, those of the 3-stage policies were not substantially better and in a few cases were worse than those of the 2-stage policies.

The sixth value-to-go approximation seemed to yield slightly better results than the other approximations. However, the third through sixth approximations were basically comparable. Overall, these approaches improved significantly upon the greedy algorithm and were able to obtain values close to the optimal for lookahead sizes greater than one. For lookahead sizes greater than one, these approaches were also able to obtain results slightly better than those obtained using rollout strategies with Monte Carlo simulations.

Table 2 The results of applying the limited lookahead policy to the three-platform problem.

Value-to-go Approximation	1-stage	2-stage	3-stage
1	491.09	539.58	543.40
2	520.57	543.95	553.74
3	506.55	550.82	553.76
4	500.69	529.98	559.46
5	554.10	557.94	561.75
6	555.97	563.45	561.09

Table 3 The results of applying the limited lookahaead policy to the four-platform problem.

Value-to-go Approximation	1-stage	2-stage	3-stage
1	543.32	574.24	582.75
2	589.56	607.31	593.08
3	581.48	613.29	615.63
4	574.06	618.55	619.45
5	582.84	594.09	606.96
6	595.44	615.10	624.16

Tables 4 and 5 provide the average values obtained over the same thousand sample trajectories by applying the pruned limited lookahead polices for lookahead sizes of two and three, using the value-to-go approximations described above. (Note that a pruned one-step lookahead policy is equivalent to the fully expanded one-step lookahead policy.) As can be seen, the results of these approaches do not vary significantly from the fully expanded lookahead policies. In some cases, the pruned policies performed one or two percent worse and in other cases, they performed one or two percent better.

Table 4 The results of applying the pruned limited lookahead policy to the three-platform problem.

Value-to-go Approximation	2-stage	3-stage
1	538.56	523.48
2	532.70	551.10
3	550.82	553.56
4	552.47	559.46
5	556.38	555.47
6	561.22	563.82

Table 5 The results of applying the pruned limited lookahead policy to the four-platform problem.

Value-to-go Approximation	2-stage	3-stage
1	573.19	575.21
2	605.57	607.21
3	608.98	616.21
4	613.23	615.38
5	595.55	592.50
6	613.49	617.04

5.2. Platform Decomposition Results

In applying platform decomposition to our problem, we considered the following approaches to ordering the platforms:

1. A single ordering in ascending order of the platform return reward.
2. All possible orderings.
3. A "rollout" of the ordering in (1) as described by Bertsekas, Tsitsiklis and Wu ([4]). I.e., assuming that the first i-1 platforms have been selected, the ith platform is determined as follows:
   i. Consider each remaining platform in turn as the next platform and leave the other vehicles in their original order.
   ii. Solve the set of single-platform problems in the given order.
   iii. Select as the ith platform that which yields the best result.

As mentioned in Section 4, there are several ways to apply the heuristic:

- The heuristic can be applied once to obtain a policy for all stages.
- The heuristic can be applied at every stage to obtain a control for the current stage using current state information.
- The heuristic can be used to generate a value-to-go approximation for a limited lookahead policy.

Table 6 provides the average values obtained over the same thousand sample trajectories by the platform decomposition approach. The result of applying the heuristic for all possible orderings and following the paths obtained for all stages is provided in the first row. The next three rows provide the results when the heuristic using the three orderings described above (least expensive to most expensive, all possible orderings, and a rollout of the orderings) is reapplied at every stage to obtain the current control. The remaining rows provide the results when the heuristic is used to provide a value-to-go approximation for a one-stage limited lookahead policy using the orderings described above is used. As can be seen, these approaches performed extremely well. The heuristic alone performed comparably to 2-stage lookahead policies, and the other variations were able to obtain strategies that yielded results that were less than one percent from the optimal expected results.

Table 6 The results of applying platform decomposition approaches. The first row provides the result of applying the heuristic for all possible platform orderings before the start of the mission and following the resulting paths. The next three rows provide the results of reapplying the heuristic at every stage using various platform orderings (1: least expensive to most expensive; 2: all possible orderings; 3: a rollout of the orderings). The last three rows provide the results of applying one-stage limited lookahead policies using the values obtained from the platform decomposition heuristic (under the various platform orderings) as an approximation to the value-to-go.

	3 platforms	4 platforms
Heuristic alone	550.85	608.89
Heuristic reapplied-1	568.81	634.97
Heuristic reapplied-2	573.83	637.81
Heuristic reapplied-3	573.83	637.81
1-stage LL-1	570.97	633.04
1-stage LL-2	571.29	635.65
1-stage LL-3	571.29	635.65

5.3 Computation Times

The following table provides the average on-line computation time (in seconds) to apply the approaches described above to one hundred sample trajectories of the four-platform problem. The off-line computation time for the limited lookahead policies was negligible. We have measured the time required to compute the controls. In practice, this time is critical since it must be within the real-time constraints of the problem. The table gives the total time to compute these controls for the ten stages. Since these times depend on the state trajectory of the system, which is random, we averaged over 100 trajectories and recorded the results in Table 7. The times for the one-stage lookahead have not been included as the time required was negligible. The experimental results were conducted on a Sun Ultra 60 workstation. As can be seen from the table, the pruned lookahead policies were significantly faster than the fully expanded lookahead policies. Considering this in combination with the fact that the performances of the two versions are comparable suggests that that pruned lookahead policies may be more useful in practice. The pruned lookahead policies were also generally much faster than the rollout algorithms using Monte Carlo simulations, whose computation times varied from 5 to over 300 seconds per sample trajectory. The decomposition approaches were extremely fast, and also provided the best results. Reapplying the decomposition heuristic at every time step appears to be the best option. However, it is not clear how easily such approaches can be applied to variations of the problem.

Table 7 Time to compute the controls for ten stages under the various approaches averaged over 100 sample trajectories of the four-platform problem. The first six lines provide the times corresponding to the fully expanded and pruned limited lookahead results given in Tables 3 and 5. The next six lines provide the times corresponding to the last six platform decomposition results given in Table 6.

	2-stage lookahead		3-stage lookahead	
	Full	Pruned	Full	Pruned
LL-1	0.77	0.12	120.4	1.8
LL-2	9.41	1.04	1358	16.4
LL-3	1.35	0.22	134.7	3.4
LL-4	6.16	0.71	716.5	9.4
LL-5	9.16	0.91	796.5	8.2
LL-6	14.85	1.73	1258	14.5
PD-Heuristic reapplied-1			0.41	
PD-Heuristic reapplied-2			9.42	
PD-Heuristic reapplied-3			1.62	
PD-1-stage LL-1			51.50	
PD-1-stage LL-2			396.52	
PD-1-stage LL-3			138.04	

6. Summary

In this paper, we have considered alternatives to using on-line simulations for approximating the value-to-go for adaptive multi-platform scheduling in a risky environment. The main limitation to using rollout algorithms with on-line simulations that was determined in

our previous effort was the amount of computation required to evaluate control options at every stage. We instead considered two alternatives.

The first approach involved examining control options over a limited horizon. In our experimental results, this method produced results that were slightly better than those obtained through rollout algorithms with on-line simulations with similar computation time. Computation time was reduced significantly by introducing a pruning technique without loss in performance.

The second approach involved decomposing the problem into sub-problems associated with each platform. This method produced results that were extremely close to the optimal values and required small computation times. However, while limited lookahead methods generalize well to other problems, the decomposition method requires a suitable problem structure. Furthermore, this method may not perform well for problems with an appropriate structure if the decomposed elements require significant coordination.

7. References

[1] Alden, J.M., Smith, R.L., "Rolling Horizon Procedures in Nonhomogeneous Markov Decision Processes," *Operations Research*, V. 40, 1992.

[2] Bertsekas, D.P., Castañon, D.A., "Rollout Algorithms for Stochastic Scheduling Problems," *Journal of Heuristics*, V. 5, 1999.

[3] Bertsekas, D.P., Castañon, D.A., Curry, M.L., Logan, D., "Adaptive Multi-platform Scheduling in a Risky Environment," *1999 Proceedings from Advances in Enterprise Control Symposium*, Nov 1999.

[4] Bersekas, D.P., Tsitsiklis, J.N., Wu, C., "Rollout Algorithms for Combinatorial Optimization," *Journal of Heuristics*, V. 3, 1997.

Approximate Dynamic Programming for Multi-Vehicle Scheduling in a Risky Environment[1]

Cynara Wu[2], Dimitri P. Bertsekas[3], David A. Castañon[4], Michael L. Curry[2], David Logan[2]

Abstract

In this paper, we investigate the use of approximate dynamic programming methods for adaptive multi-vehicle scheduling in a risky environment. We develop a mathematical formulation as a Markov decision problem, which can be solved optimally using dynamic programming, but the computational requirements are overwhelming. In this paper, we develop two approaches for generating approximate solutions. The first approach consists of using limited lookahead policies, which reduce computational requirements by considering value explicitly over a limited horizon and approximating the value of the remaining stages. This approximation can consist of analytic methods as well as simulation based methods. The second approach decomposes the problem into sub-problems corresponding to individual vehicles. We apply these approaches to two example problems to demonstrate the advantages of the various techniques. Our results indicate that the second obtains superior solutions using less computation, whereas the first approach generalizes readily to other stochastic scheduling problems.

1. Introduction

There are many important applications, such as data collection, sensor management, and vehicle routing, which require the scheduling of a limited number of vehicles to best achieve a set of mission objectives in the presence of uncertainty. Often, the information available for determining the schedule is incomplete, is not always correct, and evolves dynamically. In these situations, it is important to exploit any available real-time information to modify the planned actions in order to achieve the best performance.

In this paper, we consider the problem of scheduling multiple vehicles to perform a distributed set of tasks in a risky environment, where vehicles can be destroyed in the process of performing a task. This creates uncertainty as to which tasks will be completed. Furthermore, after losing one of the vehicles, other vehicles should be rescheduled to compensate for the missing vehicle. We formulate the above problem as a Markov decision process with full state observation. The principal approach for solving Markov decision problems is stochastic dynamic programming (DP) [2]. Unfortunately, it is well known that the computation requirements for DP are overwhelming for many important problems.

Risky multi-vehicle scheduling arises in many military applications. The work of [3, 4] formulates the dynamic routing of unmanned air vehicles as a vehicle routing problem with time windows. In a similar problem, [5] optimizes the coordinated maneuver of unmanned air vehicles using a wavelet-based optimization technique that combines evolutionary computing (e.g., neural networks) with

[1] This work was supported by the Air Force Office of Scientific Research under contract #F49620-98-C-0023.
[2] ALPHATECH, Inc., 50 Mall Road, Burlington, MA 01803.
[3] Dept of Electrical Engineering and Computer Science, M.I.T., Cambridge, MA 02139.
[4] Dept of Electrical and Computer Engineering, Boston University, Boston, MA 02215.

interior point optimization methods. These models fail to account for the inherent risk of trajectories in their planning, and do not replan trajectories when vehicles are lost.

In recent years, there has been a great deal of research on approximate DP methods [1,6-12]; these methods are collectively known as neurodynamic programming (NDP). In [9], NDP is used for vehicle routing problems with stochastic demands. In this formulation, there is no risk of losing vehicles; uncertainty arises because of unknown demand at specific centers.

In our prior work [6], we considered a class of risky scheduling problems involving a single vehicle and developed approximate DP algorithms. We extended those results to multiple vehicles [7], using approximate DP techniques based on simulation and rollout [8]. Although the results of [8] were encouraging, the computation requirements of the approximate DP algorithms were large, and unsuited for real-time rescheduling.

In this paper, we investigate classes of approximate DP algorithms for the solution of risky multi-vehicle scheduling problems where individual vehicles can be destroyed while performing tasks. In particular, we are interested in algorithms which have reduced computation requirements, while still achieving near-optimal performance. We present a model for risky multi-vehicle scheduling as a stochastic Markov decision problem with individual states evolving on a risky graph. The resulting decision problem is a Markov decision problem with a combinatorially large control space, and with temporal dynamics. We explore different approaches for approximating the future cost-to-go, as well as for searching efficiently through the combinatorial control options to determine an approximate optimal control. We evaluate the relative advantages of the different algorithms using two simulated examples.

The paper is organized as follows. In Section 2, we describe a mathematical model for risky multi-vehicle scheduling, which provides the framework for the remainder of the paper. In Section 3, we develop approximate DP algorithms based for limited lookahead policies. In Section 4, we describe a vehicle decomposition approach to obtain a different class of approximate DP algorithms. In Section 5, we discuss an important variation of the basic mathematical framework, whereby specific tasks require collaboration between multiple vehicle types. We develop new approximate DP algorithms for this class of problems. In Section 6, we present the results of our computation experiments.

2. Mathematical Formulation

In this section, we describe a mathematical formulation that represents the important aspects of multi-vehicle scheduling problem under risk. The basic paradigm we use is that vehicles are scheduled to perform spatially distributed tasks. While traveling between tasks, vehicles may be destroyed, and thus fail to reach and perform some of their assigned tasks.

Our model starts with a finite set of discrete nodes N, which represent potential locations of tasks or intermediate waypoints in routes to task locations. Each node n is characterized by a value $V(n)$ of the task associated with that node. This value can vary dynamically, and is reduced to zero if the task at that node has already been performed, or if there is no task at the node.

We assume that there is a set of directed arcs A connecting the nodes N. Associated with each of arc a is a risk $p(a)$, which represents the probability that a vehicle traversing that arc will not be destroyed on that arc, and thus will reach the end node of the arc.

Vehicles travel on the associated graph (N,A). There are M identical vehicles that are present in the graph. We describe the evolution of each vehicle in terms of a discrete time index k. We assume that each vehicle can traverse a single arc in a unit of time. Although this assumption implies that each arc can be traveled in unit time, this assumption is not restrictive because additional nodes can be introduced to represent waypoints which are equally spaced. Associated with each vehicle m at time k is a vehicle state $s_k(m)$, which is either -1 to indicate the vehicle has been destroyed, or indicates the node which contains the vehicle at time k.

The state of the system at time k is defined as x_k, and is composed of the collection of individual vehicle states and node value states. Let $v_k(n)$ denote a binary state for node n at time k, which is 1 if no vehicle has reached the node before or at time k, and 0 otherwise. Then, the state of the system is defined by the vector

$$x_k^T = [v_k(1), \ldots, v_k(N), s_k(1),\ldots,s_k(M)]^T$$

The admissible decisions at time k, denoted by $U(x_k)$, depend on the current state x_k. Each vehicle m which is not destroyed at time k (so $s_k(m) > 0$) must select an arc in A to traverse which starts at node $s_k(m)$; staying at the current node is not allowed unless there is an arc in A which starts and ends at that node. When vehicle m traverses an arc $a= (s_k(m),e)$ at time k, its state changes as follows:

$s_{k+1}(m) = e$ with probability $p(a)$; otherwise, $s_{k+1}(m) = -1$.

We assume that the stochastic events associated with each vehicle traversing an arc are independent across vehicles, arcs and time. Thus, each arc traversal represents an independent Bernoulli event that affects the state of an individual vehicle.

The node value dynamics are deterministic, and represented as follows: If a vehicle reaches a node, its node value is automatically reduced to 0. With this notation, the state dynamics can be represented as

$$x_{k+1} = f_k(x_k, u_k, \omega_k)$$

where x_k is the state, u_k is the control to be selected from a finite set $U_k(x_k)$, and ω_k represents the random events associated with the arc transitions by vehicles.

The final aspect of the model is the objective function. We formulate the decision problem as a finite horizon problem, with maximum time T. We assume that, at time $k=0$, all vehicles start at a base node $n=0$. We are interested in all vehicles returning to base by time T. Associated with each vehicle is a vehicle value V_m, which is lost if the vehicle does not safely return to base. With this notation, the net value at the final time T when the system reaches state x_T is given by the sum of task value accumulated minus vehicle value lost, as

$$J(x_T) = \sum_{n \in N} V(n) I(v_T(n) = 0) + \sum_{m=1}^{M} V_m I(s_T(m) = 0)$$

We can regroup the above performance in terms of the incremental value accumulated at each time, as follows:

$$J = \sum_{k=1}^{T} \{ \sum_{n \in N} V(n) I(v_{k-1}(n) = 1, v_k(n) = 0) \} + \sum_{m=1}^{M} V_m I(s_T(m) = 0) = \sum_{k=0}^{T-1} g_k(x_k, u_k, \omega_k) + G(x_T)$$

where $I()$ is the indicator function. The objective J is random, as it depends on the outcomes of the random arc traversal events. We assume that, at each state k, the state x_k is observed, and the choice of control action u_k depends on the current state x_k.

Although the above formulation assumes that all vehicles are identical, it is straightforward to include multiple types of vehicles, by making the arc transition probabilities depend on vehicle type j, as $p(a,j)$. It is also straightforward to make the value of a node depend on the time at which a vehicle reaches the node, as $V(n,k)$, thereby incorporating constraints such as windows of time during which tasks are available at nodes. A third straightforward extension is to restrict the type of vehicle which can perform the task at a specific node. Although we do not explicitly treat such examples in this paper, the methodology presented below extends naturally to those cases.

The above model is a Markov decision problem, with observed state x_k. Note, however, that the number of possible different states is $2^V(V+1)^M$, where V is the number of nodes in the graph. Thus, the number of states explodes rapidly with the number of nodes. This makes infeasible computation of a full feedback strategy, which selects an action for every possible state and time. In this paper, we investigate approximation techniques which do not require such extensive computations.

2.1 **Example Data Collection Problem**

The graph in Figure 1 is an example of the above model corresponding to a data collection problem. Each node represents a geographical area of interest with a one-time value denoted next to the node. The arcs represent connectivity among the geographical regions and may be successfully traversed with a known probability, indicated on each arc. Vehicles traverse the graph and collect data (value) at each node, or else they are destroyed while traversing specific arcs. If a vehicle is destroyed on an arc, the value of the destination node is not collected, which can result in retasking other vehicles

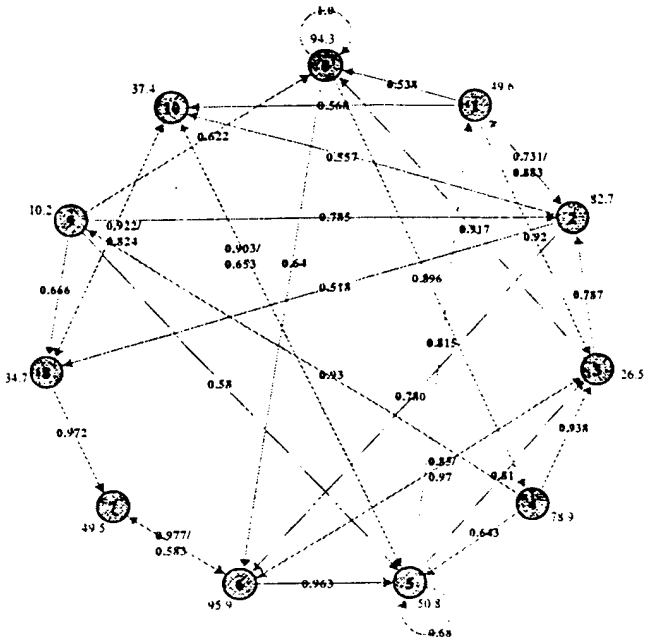

Figure 1 Graph representation of an example data collection problem.

The objective is to control the vehicles in order to maximize the expected total value collected after T stages. Each vehicle begins at a base or hub node (in this case, node 0 for all vehicles) and may traverse one arc during each stage. There is a reward for each vehicle that has safely returned to its base node at the end of the Tth stage.

3. Limited Lookahead Policies

Consider the discrete-time dynamic system model described in Section 2, as

$$x_{k+1} = f_k(x_k, u_k, \omega_k)$$

A control policy $\pi=\{\mu_0,\mu_1,\ldots,\mu_{N-1}\}$ maps, for each stage k, a state x_k to a control value $\mu_k(x_k)\in U_k(x_k)$. Denote the single stage reward as $g(x_k u_k w_k)$ and the terminal reward as $G(x_T)$. The value-to-go of an optimal policy $\pi^*=\{\mu_0^*,\mu_1^*,\ldots,\mu_{N-1}^*\}$ starting from a state x_k at stage k can be computed using the following DP recursion

$$J_k^*(x_k)=\max_{u_k\in U_k(x_k)} E\{g_k(x_k,u_k,\omega_k)+J_{k+1}^*(f_k(x_k,u_k,\omega_k))\},$$

for all k and with the initial condition $J_T^*(x_T)=G(x_T)$.

A *one-step lookahead* policy selects the control at stage k and state x_k to maximize the following expression:

$$\max_{u_k\in U_k(x_k)} E\{g_k(x_k,u_k,\omega_k)+\tilde{J}_{k+1}(f_k(x_k,u_k,\omega_k))\},$$

where $\tilde{J}_{k+1}$ is some approximation of the value-to-go function J^*_{k+1}. A *two-step lookahead* policy selects the control at stage k and state x_k to maximize the above expression when $\tilde{J}_{k+1}$ is itself a one-step lookahead approximation; i.e., for all possible states $x_{k+1} = f_k(x_k, u_k, \omega_k)$, we have

$$\tilde{J}_{k+1}(x_{k+1}) = \max_{u_{k+1} \in U_{k+1}(x_{k+1})} E\left\{g_{k+1}(x_{k+1}, u_{k+1}, \omega_{k+1}) + \tilde{J}_{k+2}(f_{k+1}(x_{k+1}, u_{k+1}, \omega_{k+1}))\right\}$$

Other multi-stage lookahead policies are similarly defined. Note that the number of lookahead stages, L, should be less than or equal to T-k-1. Essentially, the L-stage lookahead policy selects at stage k its decision by determining the optimal policy if there were only L stages remaining and the terminal cost was given by $\underset{x_L}{E}\left\{\tilde{J}_{k+L+1}(x_L)\right\}$, where x_L is the state resulting from applying the policy for the L decisions.

The main advantage of lookahead policies is that the control action can be selected by only considering those future states which can be reached in during the limited lookahead period. In terms of the model in Section 2, this limits the number of nodes that can be reached, and thus the number of states which must be considered.

When using a lookahead policy at stage k and state x_k, a decision is selected, and the process is repeated at the next stage. The lookahead horizon is limited to the number of remaining stages, and so if the number of remaining stages is less than L, the L-stage lookahead policy determines the optimal strategy. A special case of such policies in which the value-to-go is approximated with zero is referred to in the literature as rolling or receding horizon procedures [1,14]

The effectiveness of limited lookahead policies depends on two factors:

- The quality of the value-to-go approximation – performance of the policy typically improves with approximation quality.

- The length of the lookahead horizon – performance of the policy typically improves with the length of the horizon.

However, as the length of the lookahead horizon increases, the number of possible states which must be considered increases exponentially. To keep the overall computation practical, the complexity of the value-to-go approximation should be reduced for larger lookahead sizes. Balancing such tradeoffs is therefore a critical element in determining the size of the lookahead and the method for approximating the value-to-go. This paper explores several possibilities and tries to quantify the associated tradeoffs. One of the advantages of using limited lookahead policies for our particular problem is that the number of controls at a particular stage is fairly small. As a result, the computation required to explore all states that can be visited over the next L stages is manageable for small L.

Since the number of states that can be visited over L stages grows exponentially in L and also in the number of vehicles, limited lookahead policies for L>1 are impractical for problems with many

vehicles. One approach to reducing the computation required for limited lookahead policies is to limit the number of states that can be visited. This can be accomplished by removing from consideration controls that yield inferior intermediate values.

A *selective* version of a limited lookahead policy depends on an integer parameter B. We determine the one-step lookahead values for all controls available from our initial state. Controls that are among those with one of the B best one-step lookahead values are selected for further exploration. We then repeat this process for each state that can be reached from a control that was selected and determine the one-step lookahead values for all controls available from these states. For each of these states, controls that are among those with one of the B best one-step lookahead values are selected for further exploration. The number of times this process takes place is equal to the size of the lookahead.

Since the number of controls that are expanded from every state at every stage is limited, the computation required to find selective policies is not exponential in the number of vehicles. However, the computation is still exponential in the size of the lookahead.

4. Vehicle Decomposition Algorithms for Multi-vehicle Scheduling

We now present an alternative approximate DP approach that involves exploiting the structure of our specific model, by decomposing it into a set of simpler problems. In particular, we decompose the problem into a sequence of subproblems, each involving a single vehicle. The advantage of this decomposition is that the single vehicle problem reduces to a version of the quiz problem studied previously in [6]. The results in [6] establish that the optimal feedback strategy can be parameterized in terms of a deterministic sequence of nodes to visit, greatly reducing the search required for an optimal strategy.

The subproblem for a single vehicle consists of determining the optimal sequence of nodes, or path, to visit assuming that vehicle was the only one available. After a subproblem is solved for a particular vehicle and before the next subproblem is solved, the value of each node in the associated path is updated to the value of the node multiplied by the probability that the node was not visited by the vehicle. This allows vehicles to take into account paths assigned to previously scheduled vehicles. When all of the subproblems have been solved, a set of paths for each vehicle results. An outline of the vehicle decomposition approach is given below.

1 Assume that the vehicles are ordered 1,2,...,M, and start with vehicle i=1.
2 Solve the single-vehicle problem optimally by finding a path or sequence of nodes $(n_{i1}, n_{i2}, \ldots, n_{iT})$ that the vehicle should attempt to visit in order to maximize its expected value.
3 For every node in the path obtained in (2), scale the value of the node to 1 minus the probability that the node will be visited by vehicle i. This allows vehicles that are scheduled later to take into account the path assigned to the current vehicle.
4 If i is less than the number of vehicles, then let i=i+1 and go to (2). Otherwise, stop.

The single-vehicle problem in step 2 can be solved using dynamic programming or by exhaustively considering all possible paths with N nodes. The computation required in either case is $O(D^N)$, where N is the number of stages and D is the average degree of a node. For sparsely connected

graphs, the computation required is minimal. The set of subproblems can be solved once for a particular ordering of vehicles or multiple times for various vehicle orderings. We will investigate several possibilities in the numerical experiments in Section 6.

The vehicle decomposition heuristic may be applied either once, before the mission begins, but also at multiple times at different stages of the mission, taking into account only vehicles that are still alive and nodes whose value has not yet been collected. The heuristic can also be used to compute a value-to-go approximation for limited lookahead policies.

One of the main advantages to the vehicle decomposition approach is that the computation required is considerably smaller than limited lookahead policies. Assuming that the number of vehicle orderings considered remains fixed, the computation grows linearly in the number of vehicles. In addition, as will be seen below, the method obtains solutions that are very close to the optimal. Note, however, that the limited lookahead approach generalizes readily to other problems that may not have easily decomposable structures.

5. Model Extensions

In this section, we extend the basic model of Section 2 to represent additional classes of risky multi-vehicle scheduling problems. In particular, we consider extensions that require coordinated action by multiple vehicles to accomplish the required task at a node, and problems with large decision spaces.

As an example, consider the problem illustrated in Figure 2. In this problem, the three nodes at the bottom are hub nodes where vehicles are based and the eleven nodes at the top are task nodes to be visited by vehicles. There is a vehicle of Type 1 and a vehicle of Type 2 located at each of the hub nodes. At each stage, vehicles have the option of remaining at their hub node or of attempting to visit and return from one of its connected target nodes. The control at a particular stage is the set of options for all of the vehicles. Here, each vehicle has a larger number of nodes that it can visit than in the previous example. As a result, the size of the control space makes enumerating all of the controls for a single-step lookahead method impractical, let alone for multi-step lookahead methods. For the example in Figure 2, the number of possible controls at every stage if all of the vehicles are still alive is 44,100.

In this example, nodes 6 and 7 require a visit by a vehicle of Type 1 followed by a visit by a vehicle of Type 2 for its value to be collected. This precedence constraint requires an extension to our previous model. Furthermore, good performance will require a control strategy that coordinates movements among the vehicles. As a result, the vehicle decomposition methods may not be applicable to this problem.

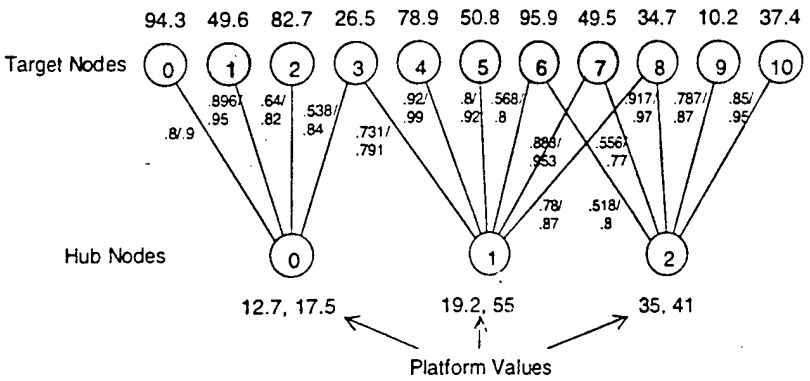

Figure 2 Example with precedence constraints. At each of the hub nodes at the bottom are based a vehicle of Type 1 and a vehicle of Type 2. The values for the two vehicles are indicated below the corresponding hub node. The values available to be collected at each of the target nodes at the top are provided above the corresponding node. Values at shaded nodes are collected when a visit by a Type 1 vehicle is followed by a visit from a Type 2 vehicle.

We address the two issues introduced above, that of an extremely large control space and that of vehicle coordination constraints, in the following subsections.

5.1 Mathematical Model of Precedence Constraints

We extend our prior model to include precedence constraints on vehicle types in order to complete the task at that node. Mathematically, we assume that there are two types of vehicles in the problem, and we assume that nodes with precedence constraints require a visit from a vehicle of type 1 before or concurrent with a visit from a vehicle of type 2 to perform the node task. Let N_1 denote the nodes with precedence constraints, and let N_2 denote the rest of the nodes. For nodes n in N_1, we modify the definition of node state as follows:

$$v_k(n) = \begin{cases} 2 \text{ if no platform has visited node } n \text{ before or up to stage } k \\ 1 \text{ if a platform of type 1 has visited node } n \text{ before or up to stage } k, \text{ but not one of type 2} \\ 0 \text{ if both platform types visited the node in the right order} \end{cases}$$

The state dynamics for nodes with precedence constraints are straightforward to define. If a vehicle of type 1 reaches the node at stage k, and its state is $v_{k-1}(n) = 2$, its state switches to $v_k(n) = 1$. If a vehicle of type 2 reaches the node at stage k, and its state is $v_{k-1}(n) = 2$, there is no state transition; if $v_{k-1}(n) = 1$, then $v_k(n) = 0$. If vehicles of both types reach the node at stage k, then $v_k(n) = 0$.

With this definition of node state, the overall problem state and objective functions remain the same as described in Section 2.

5.2. Pruned Control Space Algorithms

We now describe strategies for solution of problems with large control spaces, without enumerating all of the control options. Our approach is to consider only a limited or "pruned" set of controls. Mathematically, under a one-step lookahead pruned policy, the control selected at stage k and state x_k is selected to maximize the following expression:

$$\max_{u_k \in \overline{U}_k(x_k)} E\left\{g_k(x_k, u_k, \omega_k) + \tilde{J}_{k+1}(f_k(x_k, u_k, \omega_k))\right\},$$

where $\overline{U}_k(x_k) \subset U_k(x_k)$ is some subset of all of the possible controls at state x_k and $\tilde{J}_{k+1}$ is some approximation of the optimal value-to-go function J^*_{k+1}. The key is determining the subset of possible controls which will be evaluated. Note that this method differs from the selective lookahead approach because we remove controls from consideration without receiving any direct evaluation using approximate dynamic programming.

We propose two methods for pruning the control space, based on the solution of approximate one-step problems: a greedy approach, and an assignment approach. In the greedy approach, each vehicle independently determines the value of each of its individual options using some heuristic and selects a small number of options with the best values. The set of candidate controls $\overline{U}_k(x_k)$ at state x_k consists of all combinations of the individual best options selected for each vehicle. Although this approach eliminates a large number of controls, the number of controls remaining grows exponentially in the number of vehicles.

An alternative approach is to solve approximately an associated assignment problem. Under such an approach, we heuristically obtain values for each vehicle's options as in the greedy approach. For each vehicle, we select one candidate control corresponding to each of its individual possible options as follows. The given vehicle is assigned to one of its options and the options for the remaining vehicles are determined by solving an auxiliary assignment problem, as described below.

In the classic assignment problem, persons and objects are to be assigned to each other in a one-to-one map. Each object carries a given value to each person, and the goal is to assign the objects to persons so as to maximize the total collected value. In our assignment problem, vehicles can be viewed as persons. The objects are the nodes that a vehicle can visit in one step. Since there is one candidate control corresponding to each eligible next node for every vehicle, the number of controls which must be evaluated by approximate dynamic programming at every stage grows linearly with the number of vehicles.

In both of the approaches, the key is to associate values with each individual vehicle control option at stage k, in order to formulate single stage assignment problems. We discuss the value assignment techniques in the experimental results in Section 6.

5.3. Algorithms for Problems with Precedence Constraints

The pruning methods discussed above can be extended to problems with precedence constraints, by heuristically providing partial values for controls that satisfy in part the precedence constraints of the visited node. However, the use of one-to-one assignment models cannot incorporate the needed coupling that results when two vehicles of different types visit the same node, particularly at different times.

In this section, we propose an alternative approach for determining controls for problems with precedence constraints. The approach is based on principles from model-predictive control [14], where an optimal sequence of control actions is determined based on solving an approximate deterministic, finite-horizon problem defined from the current state. The subset of controls that correspond to actions at the current state is implemented, leading to a change in state. The process is repeated at each stage.

The key to this approach is defining a multistage deterministic scheduling problem that considers the coupling between the controls over multiple stages. We formulate this problem as follows. Consider the nodes as persons, and vehicle visits during a particular stage as objects, we want to formulate the multistage scheduling problem as a "bundling assignment" problem ([15-17]). In such a problem, persons are to be assigned to bundles consisting of subsets of objects. That is, we want to assign a set of vehicle visits over the specified time interval (i.e. a bundle) to each node. Each node has a value assigned for each bundle. Furthermore, each object (vehicle-visit time) has constraints on which bundles it can belong to, given the reachability constraints implicit in the scheduling graph. The objective is to find the feasible persons to bundles assignment that maximizes the total value. Such problems are known to be NP hard and have been the subject of considerable recent research ([15-17]). Both exact and approximate solution methods have been considered for cases without vehicle-to-bundle assignment constraints. Here we present a new approximate solution which extends to our case with vehicle-to-bundle assignment constraints, based on the rollout approach of [8]..

The bundle "rollout" algorithm allocates bundles to persons one at a time. It is an iterative technique, which loops once over the set of persons. We describe an iteration below:

- Start with a subset of persons with already assigned bundles. Select the next person without an assigned bundle.
- For this person, determine all remaining feasible bundle assignments. For each feasible bundle assignment, do:
  - Solve an auxiliary one-to-one assignment problem to allocate the remaining objects to the other unassigned persons, where individual values have been assigned to each possible one-to-one assignment. This auxiliary problem simplifies the bundle constraints, making it easy to solve.
  - Combine the auxiliary assignment solution with the feasible bundle assignment to the current person and evaluate the outcome.
- Select the feasible bundle assignment for the current person with the largest overall value, and assign it permanently to this person.

This iteration is repeated for the remaining persons and objects. As this rollout algorithm progresses, bundles are assigned one by one until all bundles are exhausted.

The bundle assignment problem is used to calculate multi-vehicle controls over a multi-stage horizon. We illustrate the algorithm for a two-stage rolling horizon. Assume that we are at state x_k at stage k. We formulate a bundle assignment problem that determines the nodes that vehicles will visit over the next two stages, k and $k+1$. In this problem, each node may be assigned to a bundle of at most two vehicles. Each vehicle can be assigned twice, once for stage k, and once for stage $k+1$. However, a vehicle at stage k can only be assigned to bundles for nodes that can be reached by the vehicle in one transition, and at stage $k+1$ to nodes which can be reached in two transitions. Thus, a bundle will consist of a pair of vehicle-stage assignments which are feasible for a node, where we allow empty vehicle-stage assignments to nodes which do not receive complete bundles. The value of that bundle is determined based on the node's state at k, its value, the vehicles' values, the stage of the assignment and the probabilities of survival of the vehicles on the arcs to reach the node. We delay the discussion of the assignment of bundle values until Section 6.

Suppose now that we have obtained an approximate solution of the kth stage bundle assignment problem. During the kth stage, our rolling horizon algorithm uses the target-to-vehicle assignment specified by the kth stage portion of this approximate solution (which covers stages k and $k+1$ as discussed above). At the next [$(k+1)$st] stage, the process is repeated using the new state x_{k+1}. Extending the above procedure to arbitrary horizon lengths is straightforward. However, a longer horizon also results in a larger number of possible bundles and an attendant increase in computation needed to solve approximately the bundle assignment problem.

6. Computational Results

We now present some computational results from applying the above approaches to the problems illustrated in Figures 1 and 2.

6.1. Example 1 Experiments

We first consider the problem illustrated in Figure 1. For this problem, we consider an instance with $N=10$ stages and four vehicles. The return rewards for the individual vehicles were set to 12.7, 17.5, 19.2, and 55.0. In this problem, there are few vehicle control options from any node, so the total number of control combinations was small, and pruning was not required. We therefore only consider the limited lookahead policies and the vehicle decomposition approaches.

6.1.1 Limited Lookahead Policies

A limited lookahead policy consists of two main elements: the lookahead horizon, and the approximation of the value-to-go. In the experiments below, we vary the size of the horizon from one to three and consider several different straightforward approximations to the value-to-go.
In many of our approaches, the value-to-go approximation for a particular state x after the first k stages, $\tilde{J}_k(x)$, involves heuristically generating for each vehicle i, a path or sequence of nodes $(n_{i(k+1)}, n_{i(k+2)}, \ldots, n_{iN})$ to attempt to visit during the remaining T-k stages. We denote this collection of paths $P(x,k)$. Assuming each vehicle attempts to visit the nodes in its path, we can determine the expected collected value resulting from visiting nodes not visited during the first k stages:

$$C[P(x,k)] = \sum_{\substack{\text{nodes } n \text{ not} \\ \text{yet visited}}} \left(1 - \prod_{\text{platforms } i} (1 - p_{in}) \right) v_k(n).$$

In the above equation, $v_k(n)$ is the value associated with node n at stage k and p_{in} is the probability that vehicle i visits node n after stage k:

$$p_{in} = \begin{cases} \prod_{j=k}^{l-1} p(n_{ij}, n_{i(j+1)}), & \text{if } n_{il} = n \text{ for some } l, \\ 0, & \text{otherwise,} \end{cases}$$

where $p(n_{ij}, n_{i(j+1)})$ is the probability of successfully traversing the arc connecting nodes n_{ij} and $n_{i(j+1)}$. The term $\prod_{\text{platforms } i} (1 - p_{in})$ provides the probability that none of the vehicles successfully visits node n.

The term $\left(1 - \prod_{\text{platforms } i} (1 - p_{in}) \right) v_k(n)$ then provides the expected collected value at node n (the probability that at least one vehicle successfully visits the node multiplied by the node value).

We can also determine the expected reward resulting from vehicles returning to the base node:

$$R[P(x,k)] = \sum_{\text{platforms } i} q_i V_i,$$

where

$$q_i = \begin{cases} \prod_{j=k}^{N-1} p(n_{ij}, n_{i(j+1)}), & \text{if } n_N \text{ is the base node,} \\ 0, & \text{otherwise,} \end{cases}$$

is the probability that vehicle i returns to the base node and V_i is the vehicle return reward.
The approximations to the value-to-go that we consider are given below. As can be seen in the descriptions, many of the approximations involve a combination of the expected collected node value, $C[P(x,k)]$, and the expected vehicle return reward, $R[P(x,k)]$, assuming each vehicle attempts to visit the nodes in the paths specified in $P(x,k)$.

1 The first approach approximates the value-to-go with zero:

$$\tilde{J}_k(x) = 0.$$

2 The second approach approximates the value-to-go with the sum of the expected collected node value and the expected vehicle return reward collected over a set of greedy paths:

$$\tilde{J}_k(x) = C[P_g(x,k)] + R[P_g(x,k)].$$

The nodes along the greedy path for vehicle i, $(n_{i(k+1)}, \ldots, n_{iv})$, are determined as follows:

$$n_{i(j+1)} = \arg \max_{n \in \eta(n_{ij})} \{ p(n_{ij}, n) \cdot v_k(n) \},$$

where $\eta(n_{ij})$ is the set of nodes that can be reached from node n_{ij}, and n_{ik} is the node at which vehicle i is located after k stages.

3 The third approach approximates the value-to-go with the expected vehicle return reward collected over the set of "safest" paths:

$$\tilde{J}_k(x)=R[P_s(x,k)].$$

The safest path is that which yields the highest probability of a vehicle returning successfully to its base node. These paths can be computed apriori using dynamic programming. (Essentially, the computation is equivalent to solving a set of shortest path problems.)

4 The fourth approach approximates the value-to-go with the sum of the expected collected node value and the expected vehicle return reward collected over the set of safest paths:

$$\tilde{J}_k(x)=C[P_s(x,k)]+R[P_s(x,k)].$$

5 The fifth approach approximates the value-to-go with the sum of the expected collected node value and the expected vehicle return reward collected over the set of "most valuable" paths:

$$\tilde{J}_k(x)=C[P_m(x,k)]+R[P_m(x,k)].$$

The most valuable path is that which yields the highest expected total value that could be attained by a single vehicle during the remaining stages assuming none of the values at any of the nodes have yet been collected. These paths can also be computed apriori using dynamic programming.

6 The sixth approach combines (4) and (5). The value-to-go is approximated with the maximum of the values determined by those approaches.

7 The seventh approach approximates the value-to-go using the performance of a base policy on Monte Carlo simulations. The single-stage version of this approach is also referred to as a *rollout algorithm*. Rollout algorithms are discussed extensively in the recent textbook [10]. In our implementation, we use the greedy policy as the base policy for the rollout. Under a greedy policy, the next node visited by each vehicle at any particular stage is that which maximizes its expected reward for that stage. The value-to-go is approximated using the average value attained during the Monte Carlo simulations. The number of simulated stages (length of the simulated planning horizon) varied from 1 to 10, and the number of Monte Carlo simulations per approximation varied from 5 to 40. This method requires significantly more computation than the preceding approximations and it is impractical to use this approximation for lookahead horizons greater than one.

Table 1 provides the optimal values for the problem illustrated in Figure 1. We have computed these values using dynamic programming, and the computation required was approximately one week on a Sun Ultra 60 workstation. Table 1 also provides the results of applying a greedy algorithm, in which each vehicle selects as its next node that which maximizes its expected collected value for that stage, to one thousand sample trajectories.

Table 1 Optimal and Greedy Performance for Figure 1 Problem.

Optimal	Greedy
641.0	533.89

Table 2 provides the values averaged over one thousand sample trajectories by applying the limited lookahead polices for lookahead sizes of one to three, using the first six value-to-go approximations described above. The particular approximation approach used is given in the leftmost column. The table also provides the range of values obtained by applying a one-stage lookahead policy using the seventh approximation when the number of Monte Carlo simulations used to evaluate the performance of the greedy policy was at least 20 and the length of the planning horizon was at least 6. These values are averages over one hundred trajectories.

As can be seen, while the 2-stage policies generally provided results that improved significantly upon those of the 1-stage policies, those of the 3-stage policies were not substantially better and in a few cases were worse than those of the 2-stage policies. The sixth value-to-go approximation seemed to yield slightly better results than the other non-simulation based approximations. However, the third through sixth approximations were basically comparable. Overall, these approaches improved significantly upon the greedy algorithm and were able to obtain values close to the optimal for lookahead sizes greater than one. The rollout policy performed significantly better than the other one-stage policies but generally under-performed the 2- and 3-stage policies combined with the third through sixth approximations. Unfortunately, it was not computationally feasible to implement a 2- or 3-stage rollout policy (the seventh approximation approach).

Table 2 Limited Lookahead Policy Results for Different Approximations, Figure 1

Value-to-go Approximation	1-stage	2-stage	3-stage
1	543.32	574.24	582.75
2	589.56	607.31	593.08
3	581.48	613.29	615.63
4	574.06	618.55	619.45
5	582.84	594.09	606.96
6	595.44	615.10	624.16
7	600-610	-	-

Table 3 provides the average values obtained over the same thousand sample trajectories by applying the selective limited lookahead polices for lookahead sizes of two and three, using the first six value-to-go approximations described above. (Note that a selective one-step lookahead policy is equivalent to the fully expanded one-step lookahead policy.) As can be seen, the results of these approaches do not vary significantly from the fully expanded lookahead policies. In some cases, the selective policies performed one or two percent worse and in other cases, they performed one or two percent better.

Table 3 Selective Limited Lookahead Policy Results, Figure 1

Value-to-go Approximation	2-stage	3-stage
1	573.19	575.21
2	605.57	607.21
3	608.98	616.21
4	613.23	615.38
5	595.55	592.50
6	613.49	617.04

The following table provides the average on-line computation time (in seconds) to apply the limited lookahead approaches to one hundred sample trajectories of the example problem. The off-line computation time for the limited lookahead policies was negligible. We have measured the time required to compute the controls. In practice, this time is critical since it must be within the real-time constraints of the problem. The table gives the total time to compute these controls for the ten stages. Since these times depend on the state trajectory of the system, which is random, we averaged over 100 trajectories and recorded the results in Table 7. The times for the one-stage lookahead have not been included as the time required was negligible for the first six value-to-go approximations. The average on-line computation time for the one-stage lookahead using Monte Carlo simulations varied from 100 to over 300 seconds per sample trajectory depending on the number of simulations used per approximation (20 to 40) and the length of horizon of the simulation (6 to 10). The experimental results were conducted on a Sun Ultra 60 workstation. As can be seen from the table, using on-line simulations to approximate the value-to-go for a 1-stage lookahead method requires significantly more computation than the 2-stage lookahead methods or 3-stage selective methods using analytic approximations. In addition, the selective lookahead policies were significantly faster than the fully expanded lookahead policies. Considering this in combination with the fact that the performances of the two versions are comparable suggests that that the selective lookahead policies may be the most useful in practice.

Table 4 Average Computation Time for Different Lookahead Algorithms

	2-stage lookahead		3-stage lookahead	
	Full	Selective	Full	Selective
LL-1	0.77	0.12	120.4	1.8
LL-2	9.41	1.04	1358	16.4
LL-3	1.35	0.22	134.7	3.4
LL-4	6.16	0.71	716.5	9.4
LL-5	9.16	0.91	796.5	8.2
LL-6	14.85	1.73	1258	14.5

6.1.2 Vehicle Decomposition Results

In applying vehicle decomposition to our problem, we ordered the vehicles to determine the sequence in which the corresponding subproblems were to be solved. In cases where there are multiple orderings, the subproblems are solved according to each ordering, and the control corresponding to

the ordering with the best resulting value is selected. We considered the following approaches to ordering the vehicles:

- A single ordering in ascending order of the vehicle return reward.
- All possible $V!$ orderings, where V is the number of vehicles.

A "rollout" of the ordering in (1) as described by Bertsekas, Tsitsiklis and Wu ([8]). I.e., assuming that the first i-1 vehicles have been selected, the ith vehicle is determined as follows:

- Consider each remaining vehicle in turn as the next vehicle and leave the other vehicles in their original order.
- Solve the set of single-vehicle problems in the given order.
- Select as the ith vehicle that which yields the best result.

This results in $1+2+\cdots+V$ orderings.

As mentioned in Section 4, there are several ways to apply the vehicle decomposition heuristic:

- The heuristic can be applied once to obtain a policy for all stages.
- The heuristic can be applied at every stage to obtain a control for the current stage using current state information.
- The heuristic can be used to generate a value-to-go approximation for a limited lookahead policy.

Table 5 provides the average values obtained over the same thousand sample trajectories by the vehicle decomposition approach. The result of applying the heuristic for all possible orderings and following the paths obtained for all stages corresponding to the ordering with the best result is provided in the first row. The next three rows provide the results when the heuristic using the three orderings described above (least expensive to most expensive, all possible orderings, and a rollout of the orderings) is reapplied at every stage to obtain the current control, that is using a rolling horizon approach. The remaining rows provide the results when the heuristic is used to provide a value-to-go approximation for a one-stage limited lookahead policy using the orderings described above. As can be seen, these approaches performed extremely well. The heuristic alone performed comparably to 2-stage lookahead policies, and the other variations were able to obtain strategies that yielded results that were less than one percent from the optimal expected results.

Table 5 Vehicle Decomposition Results, Figure 1

	Avg Value
Heuristic alone-best of all vehicle orderings	608.89
Rolling horizon-1	634.97
Rolling horizon-2	637.81
Rolling horizon-3	637.81
1-stage LL-1	633.04
1-stage LL-2	635.65
1-stage LL-3	635.65

Table 6 provides the average on-line computation time (in seconds) to apply the vehicle decomposition approaches to one hundred sample trajectories of the example problem. As can be seen, not only did such approaches provide the best results, they were also extremely fast. The rolling horizon approach where the decomposition heuristic is reapplied at every time step appears to be the best option. However, it is not clear how easily such approaches can be applied to variations of the problem.

Table 6 Computation Time for Vehicle Decomposition Algorithms, Figure 1

Approach	Avg Time(s)
PD-Rolling horizon-1	0.41
PD-Rolling horizon-2	9.42
PD-Rolling horizon-3	1.62
PD-1-stage LL-1	51.50
PD-1-stage LL-2	396.52
PD-1-stage LL-3	138.04

6.2. Example 2 Experiments

We now consider the problem illustrated in Figure 2. For this problem, we let N=5 stages and assume that there is a vehicle of Type 1 and another of Type 2 at each of the three hub nodes. The return rewards are indicated in the figure. As discussed previously, there are precedence constraints which require coordination between vehicles to collect values at nodes 6 and 7. In this example, we have large numbers of control options, so limited lookahead methods are only practical if the control space is pruned to a more moderate number of candidate controls at every stage.

6.2.1 Pruned Control Space Algorithms

We applied the two approaches for pruning the control space described in Section 5.2, the greedy approach and an approach based on an auxiliary assignment problem. Under the greedy approach, we determined for each vehicle-node pair, the expected value collected during the next stage if the given vehicle attempted to visit the given node. The two nodes with the highest expected single-stage values for each vehicle were selected as candidate targets. The option in which the vehicle remained at its hub node was also selected as a candidate option. The set of all candidate controls was the set of all combinations of these three options per vehicle.

Under the approach based on an auxiliary assignment problem, we obtained values for each vehicle-node pair and solve a number of assignment problems as described in Section 5.1. We considered the following methods for evaluating a vehicle-node pair.

1 The value of a vehicle-node pair is the expected value collected during the next stage if vehicle i attempted to visit node j. This is given by

$$\text{Value}(i,j)=p_{ij}n_{ij},$$

where p_{ij} is the probability that vehicle i successfully visits node j and n_{ij} is the value obtained if vehicle i visits node j.

2 The second method adjusts the first method by subtracting the expected value due to vehicle loss. This value is then divided by the probability that the vehicle does not successfully return to its hub node:

$$\text{Value}(i,j)=\frac{p_{ij}n_{ij}-(1-p_{ij}p_{ji})v_i}{1-p_{ij}p_{ji}},$$

where p_{ji} is the probability that vehicle i successfully returns from visiting node j and v_i is the value of vehicle i. We refer to the numerator as the expected one-stage vehicle and node value for the vehicle-node pair (i,j). Dividing by the probability that the vehicle does not successfully return has the effect of given preference to assignments with higher probabilities of success.

3 The third method adjusts the one-stage vehicle and node value by providing one-third of the node value if the node has precedence constraints and the vehicle is of Type 1:

$$\text{Value}(i,j)=p_{ij}\times(n_{ij}+1/3n_{2j})-(1-p_{ij}p_{ji})v_i,$$

where n_{2j} is the value collected if node j is later visited by a vehicle of Type 2.

4 The fourth method adjusts the one-stage vehicle and node value by a numerical value proportional to the probability that the vehicle successfully returns to its hub node and to the number of remaining stages:

$$\text{Value}(i,j)=p_{ij}\times n_{ij}-(1-p_{ij}p_{ji})v_i+p_{ij}p_{ji}sC,$$

where s is the number of remaining stages and C is some constant. This additional term makes it less likely for vehicles to take risky paths in early stages. The idea is that it may be better to visit a risky node that has high reward after first visiting less risky nodes with less reward.

We used the above value estimates in combination with the pruning techniques based on greedy and assignment methods to prune the control space for the problem illustrated in Figure 2, and used limited lookahead methods to evaluate each of the candidate controls. We considered four value-to-go approximations, identified below:

1 0value: approximates the value-to-go with 0.
2 Plat value: approximates the value-to-go with the expected collected node value and vehicle value at the end of the stage.
3 Greedy: approximates the value-to-go with the expected collected node value resulting from current control plus the expected collected node value resulting from an additional stage assuming all vehicles visit the node with the highest expected reward.
4 MC: approximates the value-to-go with the average total value obtained by applying greedy policy to 20 Monte Carlo simulations starting from resulting state.

Table 7 contains the results of the one-stage lookahead methods, and Table 8 contains the results of the two-stage lookahead methods. With the control space fully expanded, there are up to 44,100 possible first stage controls at any state. Under a greedy pruning method, there are up to 729 possible first stage controls, and under the pruning methods based on solving auxiliary assignment problems, there are up to 36 possible first stage controls. Under a one-stage lookahead method, the number of value-to-go approximations is equal to the number of first stage controls. Under a two-stage lookahead method, controls for all possible outcomes for each first stage control are considered, and so the number of value-to-go approximations increases significantly more than the square of the

number of first stage controls. By pruning the control space using the approach based on auxiliary assignment problems, it was possible to apply a two-stage lookahead method, even when Monte Carlo simulations were used to evaluate the value-to-go.

The results indicate that pruning controls using solutions to auxiliary assignment problems generate candidate controls that are all promising, and therefore, even limited lookahead methods with the simplest value-to-go approximations are very effective. Pruning controls using a greedy approach, however, did not perform well. As was seen in our results of applying limited lookahead methods to the problem in Figure 1, using Monte Carlo simulations to approximate the value-to-go outperformed other approximations. In addition, the two-stage lookahead methods generally significantly outperformed the one-stage lookahead methods.

Table 7 One-stage Lookahead with Pruned Controls Results, Figure 2.

1-Stage Lookahead	Fully Expanded	Greedy	Aux Assign: Plat/Node Val 1	Aux Assign: Plat/Node Val 2	Aux Assign: Plat/Node Val 3	Aux Assign: Plat/Node Val 4
0 value	403.6	361.0	408.9	433.6	461.0	463.5
Plat value	458.3	458.3	458.3	458.4	466.3	472.7
Greedy	488.1	444.1	462.5	477.2	476.4	486.3
20 M.C.	496.4	476.8	482.0	489.7	478.2	498.3

Table 8 Two-stage Lookahead with Pruned Controls Results, Figure 2.

2-Stage Lookahead	Aux Assign: Plat/Node Val 1	Aux Assign: Plat/Node Val 2	Aux Assign: Plat/Node Val 3	Aux Assign: Plat/Node Val 4
0 value	458.5	465.6	457.0	492.9
Plat value	491.1	493.4	493.1	513.9
Greedy	467.6	483.0	484.7	509.3
20 M.C.	495.8	497.9	490.5	520.1

Tables 9 and 10 provide the average on-line computation time (in seconds) to apply the one- and two-stage lookahead approaches with pruned controls to one hundred sample trajectories of the example problem. As can be seen, pruning using an auxiliary assignment problem allows the computation times for two-stage lookahead approaches to be manageable, even with a complex value-to-go approximation.

Table 8 Computation Time for One-stage Lookahead Policies with Pruned Controls, Figure 2

1-Stage Lookahead	Fully Expanded	Greedy	Aux Assign: Plat/Node Val 1	Aux Assign: Plat/Node Val 2	Aux Assign: Plat/Node Val 3	Aux Assign: Plat/Node Val 4
0 value	179.8	1.38	.26	.14	.23	.011
Plat value	253.2	1.96	.28	.15	.25	.012
Greedy	293.0	2.85	.43	.24	.39	.037
20 M.C.	110.5	2.14	.18	.15	.14	.173

Table 10 Computation Time for Two-stage Lookahead Policies with Pruned Controls, Figure 2

2-Stage Lookahead	Aux Assign: Plat/Node Val 1	Aux Assign: Plat/Node Val 2	Aux Assign: Plat/Node Val 3	Aux Assign: Plat/Node Val 4
0 value	43.7	25.2	41.6	4.38
Plat value	45.2	26.4	44.3	4.59
Greedy	60.5	36.0	60.1	7.99
20 M.C.	340.2	229.8	337.2	74.66

6.2.2 Vehicle Bundling Methods

We considered a simplified version of the rolling horizon bundle method described in Section 5.2. We implemented the two-stage rolling horizon approach described, but with a variation of the "rollout" solution to the bundle assignment problem. At the typical stage in our implementation, we allocate an appropriate bundle of vehicles to each node with a precedence constraint one at a time. For each of these allocations, the remaining nodes and vehicles are assigned according to the solution of an auxiliary assignment problem. The allocation yielding the best overall solution is used as the approximate solution to the current stage bundle assignment problem. Note that this approach only allows a single bundle to be allocated during any particular stage. The values assigned to each bundle/node assignment as well as to each individual vehicle/node pair can be tailored as above. We evaluated these assignments using approaches similar to those used to evaluate vehicle/node pairs for control pruning methods based on an auxiliary assignment problem. In particular, we used the following methods:

1 The first approach uses the expected collected node value upon an attempted visit by the given vehicle(s).
2 The second approach uses the expected collected node value upon an attempted visit by the given vehicle(s) plus the expected vehicle value if the vehicle did not attempt any further visits.
3 The third approach uses the value from the second approach divided by the probability that the vehicle/s does/do not successfully return to its/their hub nodes.
4 The fourth approach uses the value from the second approach and adds a term proportional to the number of remaining stages.

Table 11 contains the results of the vehicle bundling methods using the valuations above. The average on-line computation time per sample trajectory was negligible for all valuation methods. The bundling method slightly outperformed the two-stage lookahead approaches using the most effective value-to-go approximations, while requiring significantly less computation time. As was seen from the vehicle decomposition results, rolling horizon methods that exploit problem structures perform better than limited lookahead methods and require less computation time.

Table 9 Bundle Method Results, Figure 2

Valuation Approach	Avg Collected Value
1	484.7
2	519.0
3	524.0
4	503.0

7. Summary

In this paper, we have considered applying a number of approximate dynamic programming methods to adaptive multi-vehicle scheduling in a risky environment. These methods fall in two main categories: limited lookahead approaches and problem decomposition. Limited lookahead methods generalize easily to other stochastic scheduling problems but the computation required limits the number of lookahead stages and the complexity of the value-to-go approximations. Fortunately, it seems that two lookahead stages may be sufficient to obtain good solutions even with simple value-to-go approximations. In addition, pruning the control space using auxiliary assignment problems makes using two lookahead stages practical even for problems with extremely large control spaces.

For problems with suitable structures, decomposing the problem into sub-problems and then solving the sub-problems can reduce the computation required even more significantly than pruning. For our problems, the resulting solutions were also near optimal when a rolling horizon approach was applied. For problems requiring coordination so that decomposition is not feasible, other methods may be found that exploit the problem structure in other ways. When this is possible, a rolling horizon approach again appears to obtain good solutions with reasonable computational requirements.

References

1. Alden, J.M., Smith, R.L., "Rolling Horizon Procedures in Nonhomogeneous Markov Decision Processes," *Operations Research*, V. 40, 1992.
2. Bertsekas, D.P., *Dynamic Programming and Optimal Control*, Vol. I, 2nd Edition, Athena Scientific, 2000.
3. Ryan, J.L.; Bailey, T.G.; Moore, J.T.; Carlton, W.B., "Reactive Tabu Search in unmanned aerial reconnaissance simulations," Simulation Conference Proceedings, 1998.
4. O'Rourke, K.P., Bailey, T.G., Hill, R., Carlton, W.B., "Dynamic Routing of Unmanned Aerial Vehicles Using Reative Tabu Search," 67th MORS Symposium, 1999
5. Godbole, D., Samad, T., and Gopal, V., "Active Multi-Model Control for Dynamic Manuever Optimization of Unmanned Air Vehicles, " IEEE International Conference on Robotics and Automation, San Francisco, CA, 2000.
6. Bertsekas, D.P., Castañon, D.A., "Rollout Algorithms for Stochastic Scheduling Problems," *Journal of Heuristics*, V. 5, 1999.

7. Bertsekas, D.P., Castañon, D.A., Curry, M.L., Logan, D., "Adaptive Multi-platform Scheduling in a Risky Environment," *1999 Proceedings from Advances in Enterprise Control Symposium*, Nov 1999.
8. Bersekas, D.P., Tsitsiklis, J.N., Wu, C., "Rollout Algorithms for Combinatorial Optimization," *Journal of Heuristics*, V. 3, 1997.
9. Secomandi, N., "Comparing Neuro-Dynamic Programming Algorithms for the Vehicle Routing Problem with Stochastic Demands," Computers & Operations Research, V27, 2000.
10. Bertsekas, D. P. and J. Tsitsiklis, *Neuro Dynamic Programming,* Athena-Scientific, Belmont, MA, 1996.
11. Sutton, R.S., "Learning to predict by the methods of temporal differences," *Machine Learning*, Vol. 3, pp. 9-44, 1988.
12. Tesauro, G., and Galperin, G. R.," On-Line Policy Improvement Using Monte Carlo Search," unpublished report, presented at the 1996 Neural Information Processing Systems Conference, Denver, CO.
13. Barto, A. G., Bradtke, S. J., and Singh, S. P, "Learning to Act Using Real-Time Dynamic Programming," *Artificial Intelligence*, Vol. 72, pp. 81-138, 1995.
14. Mayne, D.Q, Rawlings, J.B., Rao, C.V., and Scokaert P.O.M., "Constrained Model Predictive Control: Stability and Optimality," Automatica, Vol. 36, 2000, pp. 789-814.
15. Parkes, D.C., Ungar, L.H., "Iterative Combinatorial Auctions: Theory and Practice," *Proc. 17th National Conference on Artificial Intelligence*, 2000.
16. Rothkopf, M.H., Pekec, A., Harstad, R.M., "Computationally Manageable Combinatorial Auctions," *Management Science*, V. 44, 1998.
17. Sandholm, T., "An Algorithm for Optimal Winner Determination in Combinatorial Auctions," *Proc. 11th National Conference on Artificial Intelligence*, 1993.